G. ISABEL B. N'GUYEN VAN TINH

PROGRAMMER EN LANGAGE MACHINE ET JOUER SUR ZX 81

POCHE *informatique* 20

Collection
POCHE-INFORMATIQUE

PROGRAMMER
EN LANGAGE MACHINE
ET
JOUER
SUR ZX 81

COLLECTION POCHE informatique

1 - G. ISABEL, *50 programmes pour ZX 81*
2 - P. GUEULLE, *Montages périphériques pour ZX 81*
3 - C. GALAIS, *Passeport pour Applesoft*
4 - R. BUSCH, *Passeport pour Basic*
5 - M. ROUSSELET, *Mathématiques sur ZX 81*
6 - C. GALAIS, *Passeport pour ZX 81*
7 - G. PROBST, *50 programmes pour Casio FX-702 et 801 P*
8 - G. PROBST, *60 programmes pour Casio PB-100*
9 - M. SAAL, *Utilitaires pour ZX 81*
10 - C. GALAIS, *Passeport pour Commodore 64*
11 - D. RANC, *Assembleur du TRS 80*
12 - D. LASSERAN, *30 programmes pour Commodore 64*
13 - G. ISABEL, *Du ZX 81 au Spectrum, 25 programmes*
14 - P. MELUSSON, *Initiation à la micro-informatique, le microprocesseur*
15 - G. PROBST, *40 programmes pour Casio PB-700*
16 - C. GALAIS, *Passeport pour Basic TO 7 et TO 7-70*
17 - D. LASSERAN, *35 programmes pour Oric 1 et Atmos*
18 - G. PROBST, *40 programmes pour Canon X-07*
19 - P. MANGIN, *Jeu sur Commodore 64*
20 - G. ISABEL et B. NGUYEN VAN TINH, *Programmer en langage machine et jouer sur ZX 81*
21 - D. LASSERAN, *30 programmes pour TO 7 et TO 7-70*

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que « les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'Art. 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les Art. 425 et suivants du Code pénal. »

© 1985 — E.T.S.F.

ISBN 2-85535-110-3 ISSN 0757-6730

Guy ISABEL
et
Bernard NGUYEN VAN TINH

PROGRAMMER
EN LANGAGE MACHINE
ET
JOUER
SUR ZX 81

Diffusion :
ÉDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue, 75940 PARIS CEDEX 19

De G. ISABEL, à ETSF

Collection* POCHE *informatique

50 programmes pour ZX 81 (nº 1)
Du ZX 81 au Spectrum, 25 programmes (nº 13)

Sommaire

Introduction 11

Première partie : *Initiation* 13

L'affichage 15
Introduction au langage machine 26
Instructions de base
Exemples

Deuxième partie : *Programmes* 55

Zig-Zag 57
Chronomètre digital géant 67
Dédale 81
Flipper 89
Casse-briques 95
Défense spatiale 107

Aide-mémoire 115

Il existe une cassette qui regroupe les 6 jeux présentés dans cet ouvrage. Si vous désirez savoir comment vous la procurer, veuillez envoyer une enveloppe timbrée à cette adresse :

Monsieur Guy ISABEL
Boîte postale 14
49550 VILLEDIEU

Les auteurs

Cet ouvrage s'adresse à tous les possesseurs de ZX 81 muni de la mémoire 16 K, et déjà familiarisés avec le langage Basic.

Or, il est possible de programmer sur cette machine à l'aide d'un langage bien plus performant en raison de son extrême rapidité : il s'agit du ***langage machine.***

Notre but est de permettre à ceux qui n'ont encore aucune notion de ce langage de parvenir à acquérir les principales connaissances nécessaires à sa programmation.

De nombreux petits programmes en langage machine introduiront très progressivement les instructions de base ; celles-ci seront largement explicitées et, très vite, vous pourrez à votre tour créer vos propres programmes en langage machine.

A la fin de l'ouvrage, 6 programmes plus sophistiqués vous permettront à la fois de vous détendre et d'apprécier pleinement toute la virtuosité du ZX 81.

Introduction

Le Basic est un langage évolué, particulièrement bien adapté à l'apprentissage de la programmation sur micro-ordinateurs. Si cet avantage est appréciable pour tous les utilisateurs du ZX 81 ou d'autres machines similaires, il n'en reste pas moins vrai que Basic n'est pas un langage rapide.

Et pourtant le microprocesseur Z 80 qui anime votre machine est capable de réaliser des prouesses de vitesse à condition de communiquer avec lui dans un langage bien particulier appelé langage machine.

Le domaine qui nous intéresse plus précisément est celui de l'affichage rapide et de l'animation ; il permet, en effet, de bien mettre en évidence la supériorité du langage machine, notamment dans la programmation des logiciels consacrés aux jeux vidéo. Les autres applications de ce langage en robotique industrielle, par exemple, ne seront pas abordées ici.

Il n'est pourtant pas dans notre intention de dédaigner les possibilités du Basic, notamment pour toutes les applications de calcul, avec la panoplie très complète des fonctions mathématiques. Nous y aurons même recours et nous n'hésiterons pas à mêler le Basic au langage machine dans notre recherche d'une plus grande efficacité.

La programmation en langage machine consiste à introduire en mémoire des codes que le processeur lit, reconnaît et exécute fidèlement (nous sommes même tentés de dire bêtement). A nous de choisir la succession des codes qui donnent le résultat escompté. Il faut encore savoir que le langage machine fait payer fort cher son efficacité en ne tolérant aucune erreur sous peine de bloquer tout le système.

Il ne reste plus alors qu'à débrancher l'alimentation.

Cette programmation périlleuse est certainement en partie responsable de l'appréhension quasi-maladive que manifestent de nombreux utilisateurs du ZX 81 à l'égard du langage machine.

Cet ouvrage devrait permettre au lecteur d'accéder enfin à un domaine passionnant et jusqu'alors réservé aux seuls professionnels de l'informatique.

Sans autre préambule, voici un premier programme qu'il vous appartient de taper pour vous faire une petite idée des possibilités extraordinaires de ce langage sur le ZX 81.

```
16514      CODE N.1       42
16515      CODE N.2       12
16516      CODE N.3       64
16517      CODE N.4       6
16518      CODE N.5       224
16519      CODE N.6       35
16520      CODE N.7       126
16521      CODE N.8       254
16522      CODE N.9       118
16523      CODE N.10      40
16524      CODE N.11      250
16525      CODE N.12      54
16526      CODE N.13      38
16527      CODE N.14      16
16528      CODE N.15      246
16529      CODE N.16      201
```

```
 1 REM ABCDEFGHIJKLMNOP
10 FOR I=1 TO 16
20 PRINT "ENTREZ LE CODE N.";I
;" ";
30 INPUT C
40 PRINT C
50 POKE I+16513,C
60 NEXT I
70 CLS
80 RAND USR 16514
90 GOTO 70
```

Qu'en pensez-vous ? Impressionnant, non ?

Si vous voulez en savoir davantage, suivez-nous et tournez les pages...

PREMIÈRE PARTIE

initiation

L'AFFICHAGE

Comme vous le savez déjà, il est très facile d'afficher sur l'écran une lettre, un chiffre, un mot ou un point, en somme l'un des caractères dont dispose le ZX 81 ou tout autre micro-ordinateur digne de ce nom.

Nous disposons d'instructions tel que PRINT, PRINT AT, PRINT TAB ou encore PLOT (pour les graphiques surtout).

Mais par ailleurs, l'écran représente d'une certaine manière le **contenu** d'une portion de la mémoire RAM (cf page 171 du manuel Sinclair) sous le nom de fichier d'affichage. Donc, en modifiant ce contenu, nous allons également modifier l'affichage.

• Qu'est-ce qu'une mémoire ?

On pourrait la comparer à un ensemble de « boîtes » juxtaposées et numérotées dans l'ordre croissant. Pour retrouver l'une d'entre elles, il suffira de connaître son numéro, c'est-à-dire son adresse.

• Que contiennent ces boîtes ?

Il faut savoir que l'ordinateur ne peut stocker dans ses circuits électroniques que des informations binaires, c'est-à-dire présentant un état 1 ou Ø.

Or, chaque boîte contient 8 informations binaires (ou bits) qui forment un **octet.**

Avec 4 bits, on peut obtenir 16 combinaisons différentes, d'où l'intérêt du système de numération à base 16 ou **hexadécimal.**

décimal	hexadécimal	binaire
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

En associant les 2 blocs de 4 bits, nous pourrons atteindre la valeur maximale hexadécimale FF.

1111	1111
F	F

Cette nouvelle configuration permet d'obtenir cette fois-ci 256 combinaisons différentes, numérotées de 0 à 255.

```
 5 REM DECIMAL EN HEXA
10 LET A$="0123456789ABCDEF"
20 FOR N=0 TO 255
25 SCROLL
30 LET Q=INT (N/16)
35 LET Q$=A$(Q+1 TO Q+1)
40 LET R=N-16*Q
45 LET R$=A$(R+1)
50 PRINT N,Q$;R$
60 NEXT N
```

0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
10	0A
11	0B
12	0C
13	0D
14	0E
15	0F
16	10
17	11
18	12
19	13
20	14
⋮	⋮
234	EA
235	EB
236	EC
237	ED
238	EE
239	EF
240	F0
241	F1
242	F2
243	F3
244	F4
245	F5
246	F6
247	F7
248	F8
249	F9
250	FA
251	FB
252	FC
253	FD
254	FE
255	FF

Pour de plus amples renseignements, nous vous conseillons de consulter le manuel du ZX 81 au chapitre 24.

Reprenons nos boîtes.

En imaginant qu'elle soient transparentes, on peut évidemment en lire le contenu. Il suffit pour cela de connaître l'adresse de la boîte qui nous intéresse. Soyons curieux à l'égard de la boîte figurant à

l'adresse 16436 ; son contenu est sans arrêt décrémenté 50 fois par seconde (variable système FRAMES). Pour connaître cette valeur, Basic dispose de l'instruction PEEK qui permet justement de lire cette valeur et de l'utiliser dans un programme.

```
10 REM CONTENU DE 16436
20 SCROLL
30 PRINT PEEK 16436
40 GOTO 20
```

Nous pourrions être tentés de modifier le contenu de l'une ou plusieurs de nos boîtes ; d'ailleurs il existe pour cela l'instruction POKE qui permet de le faire. A cet instant, il faut préciser qu'il existe en fait deux types de mémoire : tout d'abord la ROM (en anglais Read Only Memory), c'est-à-dire la mémoire qui ne se laisse que lire. Elle contient d'une manière inaltérable et permanente tous les codes nécessaires au bon fonctionnement du ZX 81.

On ne peut donc pas espérer la modifier par POKE.

Une grande partie des codes contenus dans la ROM forment des routines, c'est-à-dire en fait des sous-programmes en langage machine.

Bien que l'utilisation de certaines de ces routines soit particulièrement agréable ou pratique, nous éviterons d'y avoir recours. D'ailleurs cette façon de faire ne permettrait pas de comprendre le langage machine comme nous le souhaitions au départ.

Ensuite, il y a la RAM (en anglais Random Acces Memory). Celle-ci n'est ni permanente, ni inaltérable. En effet, elle perd son contenu au moindre défaut d'alimentation et surtout il est facile de modifier son contenu à l'aide de POKE.

Les adresses de la ROM se situent entre Ø et 8191, celles de la RAM s'échelonnent entre 16384 et 32767 avec l'extension mémoire 16 K.

Pour nous résumer, toutes les boîtes sont bien transparentes, donc lisibles mais nous dirons que celles de la ROM sont scellées, tandis qu'on peut parfaitement ouvrir celles de la RAM et en modifier le contenu.

Tapez scrupuleusement ces 4 lignes.

```
 1 REM
10 PRINT PEEK 16513
20 POKE 16513,245
30 PRINT PEEK 16513
```

Faites RUN.
La ligne 1Ø imprime le contenu de l'adresse 16513.
La ligne 2Ø le modifie, puis la ligne 3Ø le lit à nouveau.

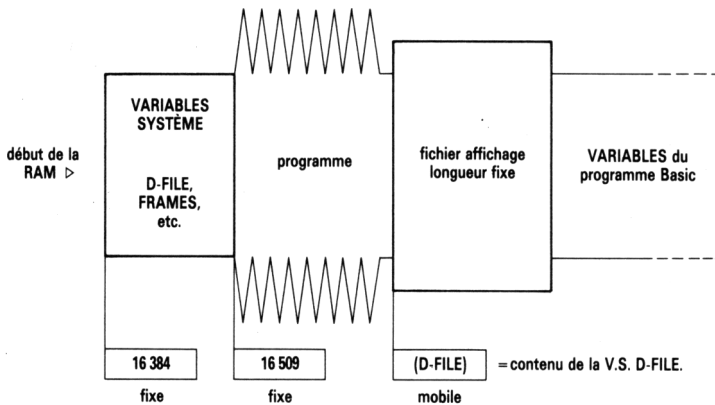
Faites LIST à présent, et trouvez l'erreur !

```
 1 PRINT
10 PRINT PEEK 16513
20 POKE 16513,245
30 PRINT PEEK 16513
```

Effectivement, PRINT vient de remplacer REM puisque le code 245 s'est substitué à 234.

Après avoir étudié succinctement les diverses boîtes qui constituent la mémoire du micro-ordinateur, il reste à aborder l'organisation de cette dernière et plus particulièrement celle de la RAM.

Le petit tableau suivant nous y aidera :



La mémoire RAM débute par une zone réservée aux **variables système.** Ces variables, toutes affectées d'un nom, sont stockées

à des adresses **fixes** de la mémoire. Elles contiennent divers renseignements indispensables au bon fonctionnement du ZX et certaines sont intéressantes à connaître pour le programmeur.

Ces renseignements sont des nombres dont quelques-uns doivent pouvoir dépasser la valeur décimale 255. Dans ce cas, nos boîtes ne pourront contenir un tel nombre. Il a fallu décomposer les plus grands d'entre eux d'une certaine manière et les ranger dans 2 boîtes successives.

Le nombre 256, par exemple, peut se décomposer en Ø plus 1 fois 256 ; donc, en stockant les chiffres Ø et 1 dans des boîtes successives et dans cet ordre, car il a son importance, nous saurons reconstituer le nombre initial.

En effet, il suffit d'ajouter le nombre contenu dans la première boîte et 256 fois le nombre contenu dans la seconde. On remarque tout de suite que cette dernière revêt une importance plus grande que la première, un peu comme le chiffre des dizaines par rapport à celui des unités.

Cette deuxième boîte contient donc l'octet le plus significatif encore appelé octet de poids fort. Le premier se nomme octet le moins significatif ou octet de poids faible.

octet 1		octet 2	
Ø		1	
adresse N		adresse N+1	
Ø	+	256 fois 1	=256

Le petit programme Basic qui suit vous aidera à découvrir de quelle manière l'ordinateur décompose les nombres, sachant par ailleurs que sur 2 octets nous ne pouvons pas représenter de nombre supérieur à 65535.

```
 5 REM DECOMPOSITION
10 SCROLL
20 PRINT "DECIMAL        OMS
OPS"
25 SCROLL
30 FOR I=Ø TO 65535
35 SCROLL
40 PRINT I;
```

```
  45 PRINT TAB 13;I-256*INT (I/2
56);
  50 PRINT TAB 22;INT (I/256)
  60 NEXT I
```

246	246	0
247	247	0
248	248	0
249	249	0
250	250	0
251	251	0
252	252	0
253	253	0
254	254	0
255	255	0
256	0	1
257	1	1
258	2	1
259	3	1
260	4	1
!	;	;
'	'	'
'	'	'
.	'	'
!	'	'
30716	252	119
30717	253	119
30718	254	119
30719	255	119
30720	0	120
30721	1	120
30722	2	120
30723	3	120
30724	4	120
30725	5	120
30726		

Le nombre 30725 s'écrit bien 5 + (256 fois 120). Vous ne serez donc plus surpris de voir des nombres stockés sur 2 adresses.

Pour décomposer un nombre quelconque N sur 16 bits, nous procédons de la manière suivante :

octet le moins significatif = N - (256 × INT (N/256))
octet le plus significatif = INT (N/256)
Prenons un exemple : soit le nombre N = 21397
OMS = 21397- (256 × INT (21 397/256)) = 149
OPS = INT (21397/256) = 83
Les codes successifs seront : 149, 83.

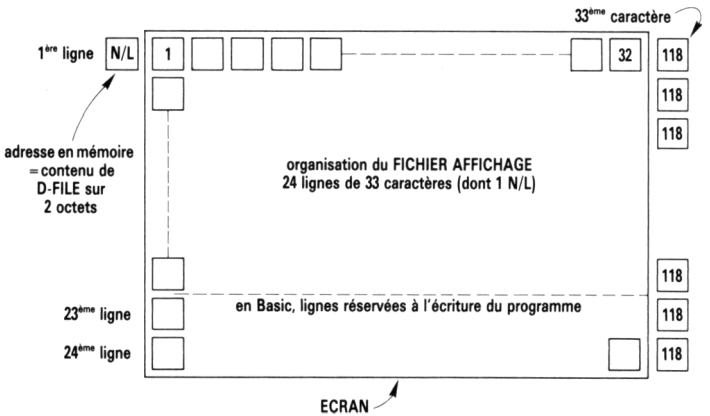
La zone mémoire réservée au programme Basic entré par l'utilisateur débute **toujours** à la même adresse, c'est-à-dire que la boîte qui contient le premier octet de cette zone se situera à l'adresse 16509.

Comme le programme peut être plus ou moins long, cette zone en mémoire occupera de même plus ou moins de place.

D'autre part, puisque le fichier d'affichage, qui n'est lui-même qu'une zone mémoire, se situe immédiatement après le programme, il se trouvera sans cesse déplacé dans sa totalité. Nous allons l'étudier davantage et retrouver plus précisément l'objet de ce premier chapitre.

Le fichier d'affichage contient tous les codes des caractères affichés sur l'écran qui toutefois ne restitue pas totalement la véritable constitution de cette zone en mémoire.

Voyons pourquoi à l'aide du dessin suivant :



On voit bien que chacune des 24 lignes sur l'écran comporte en fait 33 caractères, dont le 33e correspond à NEW/LINE soit le code 118, non visible en réalité sur l'écran, mais figurant bien dans la zone mémoire.

Le premier octet du fichier d'affichage est également un 118 qui n'est pas non plus sur l'écran.

Attention : les deux lignes du bas seront normalement réservées pour l'écriture ou l'édition du programme. Nous verrons que l'on peut tout de même y accéder facilement et les utiliser.

Puisque nous savons que la zone d'affichage se déplace en fonction de la longueur du programme, comment savoir où elle se situe en mémoire, c'est-à-dire comment connaître l'adresse de son premier octet, en l'occurence le code 118 précédant celle du premier caractère affiché ?

La réponse se trouvera **toujours** à l'adresse 16396 (et bien sûr 16397) c'est-à-dire dans la variable système D-FILE.

```
   1 REM RECHERCHE DE L ADRESSE
DU FICHIER D AFFICHAGE
  10 LET AF=PEEK 16396+256*PEEK
16397
  20 REM IMPRESSION ALEATOIRE DE
$ 32 PREMIERS CARACTERES
  30 POKE AF+1+RND*31,RND*62
  40 GOTO 30
```

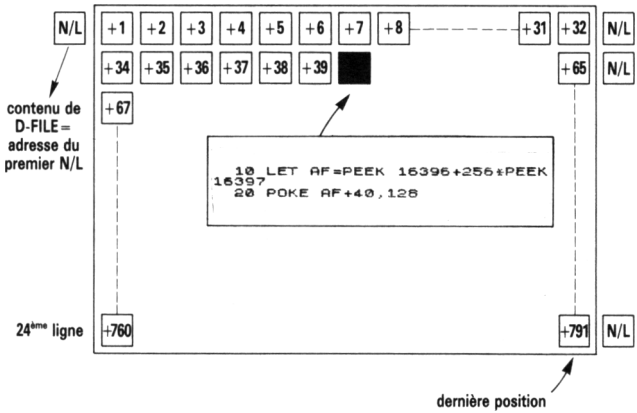
Il est possible de faire beaucoup mieux avec l'impression aléatoire de 32 premiers caractères sur l'écran.

```
   1 REM RECHERCHE DE L ADRESSE
DU FICHIER D AFFICHAGE
  10 LET AF=PEEK 16396+256*PEEK
16397
  20 REM IMPRESSION ALEATOIRE DU
 PREMIER CARACTERE
  30 POKE AF+1,RND*62
  40 GOTO 30
```

Cela consiste (ligne 30 des programmes précédents) à mettre dans le fichier d'affichage qui se trouve en mémoire, des codes aléatoires se traduisant sur l'écran par l'affichage des caractères correspondants.

⚠ Insistons encore sur le fait qu'il est impératif de ne pas toucher aux codes 118 du fichier, sous peine de voir l'ordinateur se planter.

Pour vous aider à définir n'importe quelle position sur l'écran, nous vous suggérons d'utiliser le petit tableau suivant qui permet de retrouver aisément la position du premier caractère de chacune des 24 lignes :



Si vous voulez vous amuser à utiliser POKE sur l'écran, testez ces lignes ; vous risquez au cours de vos essais de tomber sur un code 118 qui perturbera la machine.

```
  10 LET AF=PEEK 16396+256*PEEK
16397
  20 PRINT "DONNEZ MOI UN NOMBRE
< 792"
  30 INPUT N
  40 POKE AF+N,128
  50 GOTO 30
```

Autre exemple : Pour obtenir sur l'écran le déplacement d'un caractère choisi, nous utilisons deux fois l'instruction POKE à la même adresse : la première fois pour afficher un caractère, la seconde pour l'effacer (code espace = Ø).

```
   5 LET AF=PEEK 16396+256*PEEK
16397
  10 FOR I=1 TO 12
  20 FOR J=1 TO 32
  50 POKE AF+J,18
  55 POKE AF+J,0
  60 NEXT J
```

```
  70 LET AF=AF+66
  80 FOR J=1 TO 32
  90 POKE AF-J,19
  95 POKE AF-J,0
 110 NEXT J
 120 IF I=11 THEN PRINT AT 21,0;
"2 LIGNES DE PLUS"
 130 NEXT I
 140 LET K=SIN SIN SIN SIN PI
```

La ligne 140 permet de réaliser une petite temporisation sans altération de l'image.

Nous venons de découvrir le second mode d'affichage sur l'écran.

La compréhension des instructions PEEK et POKE est primordiale ; elle possède d'une certaine manière son équivalent dans le troisième mode d'affichage que nous allons découvrir à présent, mode plus performant, c'est-à-dire plus rapide, puisque réalisé en langage machine.

```
  10 REM TEST
  20 PRINT "AVEZ-VOUS BIEN COMPR
IS ?"
  30 INPUT R$
  40 IF (R$="OUI" OR R$="YES") T
HEN GOTO CHAPITRE SUIVANT
  50 GOTO CHAPITRE PEEK - POKE
```

INTRODUCTION

au langage machine

Il consiste en une suite d'instruction codées, **directement** compréhensibles et exécutées par le micro-ordinateur, contrairement au Basic qui est un langage évolué proche de l'anglais, mais qui en fin de compte doit lui aussi être traduit en codes avant d'être effectué par la machine.

Ces codes figurent tous dans la ROM.

Il est hors de question dans cet ouvrage d'étudier toutes les instructions codées du Z 80, le microprocesseur qui pilote notre ordinateur ; en effet, il en existe près de 700 différentes.

Nous verrons que la connaissance de quelques-unes d'entre elles suffira largement à nos premiers besoins.

Lorsque les codes correspondants aux instructions à réaliser sont définis, il reste encore à les implanter en mémoire, puis à lancer le programme en langage machine par l'instruction Basic USR ; (qui appelle un sous-programme comme GOSUB). USR sera évidemment suivi de l'adresse d'implantation.

Une des méthodes d'implantation des codes qui nous semble la plus simple pour les débutants, consiste à réserver en mémoire une place suffisante à l'aide de l'instruction REM suivie d'un nombre suffisant de caractères **quelconques,** destinés à être remplacés plus tard par tous les codes du langage machine.

Nous implanterons cette ligne en tête d'un programme par :

```
1 REM XXXXXXXXXXXXXXXXXXXXXXXX
```

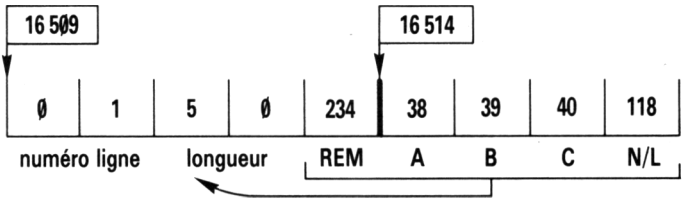
Les codes se trouveront logés à une adresse FIXE de la zone mémoire réservée au programme ; nous savons d'ailleurs qu'elle débute à l'adresse 16509.

En outre, l'adresse du premier octet réservé par REM ou premier code en langage machine est égale à 16514. Essayons de l'expliquer :

```
 1 REM ABC
10 FOR I=16509 TO 16517
20 PRINT I,PEEK I
30 NEXT I
```

```
16509          0
16510          1
16511          5
16512          0
16513          234
16514          38
16515          39
16516          40
16517          118
```

Le dessin suivant achèvera de nous préciser sous quelle forme est rangée la ligne REM en mémoire.



Les adresses 16509 et 16510 contiennent le numéro de la ligne, soit respectivement l'octet de poids fort, puis l'octet de poids faible. Les adresses 16511 et 16512 contiennent, elles, la longueur de la ligne en octets, à partir de REM et jusqu'à NEW-LINE qui valide la ligne, mais dans l'ordre inverse cette fois-ci, c'est-à-dire OPS puis OMS (c'est dans cet ordre que nous trouverons tous les nombres sous 16 bits).

Puis, on trouve les codes de REM, A (à l'adresse 16514 exactement) B, C et enfin 118 qui représente N/L.

Parlons un peu du microprocesseur Z 80.
Il possède 24 registres assimilés à des mémoires internes. Nous en utiliserons principalement 8 qui sont laissés à notre disposition par le ZX 81. Il en utilise également quelques-uns, notamment pour l'affichage.

On trouve tout d'abord le registre A encore appellé accumulateur, dans lequel nous pouvons effectuer bon nombre d'opérations. Il y a également le registre F (comme FLAG = drapeau) un peu particulier et qui contient uniquement des indicateurs binaires ; seuls les indicateurs C et Z nous seront utiles.

Nous nous servirons, en outre, des registres B et C, D et E puis H et L. Ils contiennent tous un octet, c'est-à-dire 8 bits.

Il est possible toutefois de créer les paires de registres BC, DE et HL sur 16 bits, chaque fois qu'il sera nécessaire d'y ranger un nombre ou une adresse supérieure à 255 (rappellez-vous OMS et OPS).

Nous allons enfin écrire notre premier programme en langage machine.

Exemple 1

Tapez ces quelques lignes.

```
 1 REM ABCDEFG
10 POKE 16514,42
20 POKE 16515,12
30 POKE 16516,64
40 POKE 16517,35
50 POKE 16518,54
60 POKE 16519,126
70 POKE 16520,201
```

On retrouve l'instruction POKE qui permet d'introduire tous les codes aux emplacements souhaités en mémoire.

Faites RUN puis NEW-LINE.

Les lignes 20 à 70 sont désormais inutiles et peuvent être supprimées ; nous nous apercevons que la ligne 1 s'est modifiée, son contenu « farfelu » correspond tout simplement aux caractères ayant pour code les nombres précédemment introduits.

```
1 REM E£RND7Q▓TAN
```

Rajoutez la ligne 1Ø (**Attention,** la fonction USR s'obtient à l'aide du curseur F). Puis faites RUN.

```
 1 REM E£RND7Q▓TAN
10 RAND USR 16514
```

Vous verrez un carré noir apparaître dans la première position visible de la mémoire d'écran.

C'est tout ?

Vous êtes vraisemblablement déçus, mais n'oubliez pas que nous n'avons utilisé ni PRINT ni POKE pour le faire.

En réalité, il est impossible de faire un programme en langage machine à l'aide seulement des codes numériques tels que ceux introduits à l'instant. Pratiquement, cette programmation s'opère d'abord en langage d'assemblage, qui consiste en une traduction mnémotechnique des codes eux-mêmes, codes qu'il est facile ensuite d'obtenir.

La liste complète des mnémoniques en langage d'assemblage du Z 80 et de leurs codes est donnée à l'annexe A du manuel.

A titre d'exemple, pour charger dans l'accumulateur la valeur 128, on dispose de
LD A, 128 qui se code 62, 128

Passons au langage d'assemblage du programme précédent.
LD HL, (16396)
INC HL
LD (HL), 128
RET
On remarque que ce langage ne comporte pas de numéros de lignes.

On appelle assemblage l'opération qui consiste à traduire ces mnémoniques en codes machine, ensuite exécutés dans l'ordre par le microprocesseur.

Quel sera notre raisonnement pour élaborer un tel programme ? Il s'agit très simplement de faire apparaître sur l'écran un caractère de notre choix.

Rappel : Nous savons déjà que l'écran n'est après tout qu'une zone de la mémoire que l'on nomme fichier d'affichage. En modifiant le contenu de cette zone, nous modifions l'affichage lui-même.

Dans notre exemple, nous voulons afficher l'espace inverse ayant pour code 128, au premier emplacement visible du fichier d'affichage (voir tableaux précédents). Il suffira donc de stocker le code 128 dans l'adresse de ce premier emplacement ; c'est exactement cette méthode que nous adoptions avec PEEK et POKE.

• Algorithme

Envoyer au sous-programme machine à l'aide de USR.

1) retrouver l'adresse du premier octet du fichier affichage (contient un caractère 118 = N/L).

2) l'incrémenter.

3) charger le code 128 dans cette nouvelle adresse, c'est-à-dire l'afficher.

4) retourner au Basic.

Quelles sont les instructions en langage d'assemblage qui permettent d'effectuer les différentes étapes ?

• Développement

1) Nous savons déjà que la variable système D-FILE contient sur 2 octets aux adresses 16396 et 16397 l'adresse recherchée. Chargeons son contenu dans la paire des registres HL.
Cela s'écrit LD HL, (16396)

Remarques : les parenthèses signifient que c'est bien le **contenu** de l'adresse 16396 qui est chargé et non PAS la valeur 16396.
L'écriture LD HL, (16396) charge automatiquement dans la paire HL le contenu de
16396 + 256 fois le contenu de 16397.
Cela s'explique par le fait que HL attend un résultat sur 16 bits soit 2 octets et précisément D-FILE délivre une adresse sur 2 octets.

A présent, HL prend la valeur de l'adresse du premier octet du fichier d'affichage ; cette adresse contient le code 118 ou N/L.

2) Pour obtenir la première adresse dont le contenu est affiché, il faut incrémenter HL, c'est-à-dire faire HL = HL + 1.
Nous disposons d'une instruction qui s'écrit :
INC HL

3) HL étant maintenant l'adresse du premier caractère affiché, en mettant la valeur 128 dans cette adresse, c'est-à-dire en remplaçant le contenu de HL par 128, nous provoquons l'affichage du carré noir sur l'écran.
Nous utiliserons l'instruction
LD (HL), 128.
Ici également les parenthèses signifient **contenu** de HL.
En d'autres termes, HL garde sa valeur (adresse du premier caractère sur l'écran) et son contenu (code du caractère affiché) est mis à 128.

4) Retour au Basic par l'instruction RET.
L'assemblage consiste à trouver et à ordonner les codes correspondants aux mnémoniques.

LD HL, (nn) est codé 42, OMS, OPS
adresse nn
soit pour LD HL, (16396) 42, 12, 64
(16 396 = 12 + 256. 64).
INC HL est codé 35
LD (HL), n est codé 54, n (n représentant le caractère affiché)
soit pour nous LD (HL), 128 est codé 54, 128
RET se code simplement 201

Il nous reste à présent à implanter, et dans l'ordre, dans la mémoire les codes ainsi obtenus :
soit 42, 12, 64, 35, 54, 128, 201
Il faut réserver en mémoire au moins autant d'octets que de codes à l'aide de l'instruction Basic REM suivie d'au moins 7 caractères quelconques.

```
1 REM ABCDEFGINUTILE
```

Puis à l'aide de quelques POKE nous chargeons les codes à partir de l'adresse 16514 :

```
 1 REM ABCDEFGINUTILE
10 POKE 16514,42
20 POKE 16515,12
30 POKE 16516,64
40 POKE 16517,35
50 POKE 16518,54
60 POKE 16519,128
70 POKE 16520,201
```

Faites RUN et NEW-LINE pour obtenir :

```
 1 REM E£RND70■TAN INUTILE
10 POKE 16514,42
20 POKE 16515,12
30 POKE 16516,64
40 POKE 16517,35
50 POKE 16518,54
60 POKE 16519,128
70 POKE 16520,201
```

Vous venez de comprendre comment stocker les codes en mémoire, et comment cela se traduit dans la REM.
Les lignes 10 à 70 peuvent maintenant être supprimées.
Pour lancer ce sous-programme, il suffit de préciser à la machine l'adresse d'implantation du premier code, soit l'octet 16514.
Nous obtenons à présent :

```
 1 REM E£RND70■TAN INUTILE
10 RAND USR 16514
```

A titre d'information, nous vous donnons le programme Basic correspondant :

```
  10 LET HL=PEEK 16396+256*PEEK
16397
  20 LET HL=HL+1
  30 POKE HL,128
  40 RETURN
```

Nous allons vous proposer quelques autres programmes en langage d'assemblage, introduisant peu à peu de nouvelles instructions et utilisant celles précédemment étudiées.

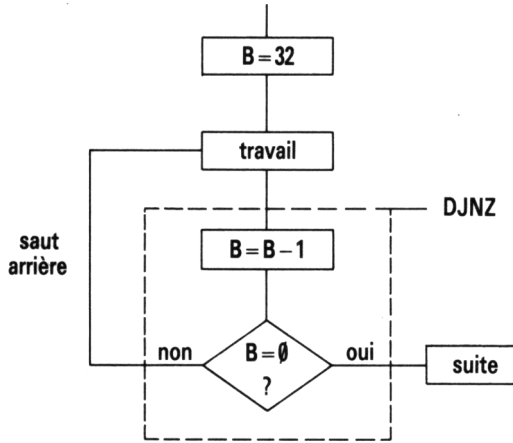
Exemple 2

Nous allons tenter d'imprimer une ligne entière de caractères 128, c'est-à-dire 32 carrés noirs juxtaposés. Pour le faire, il suffira de boucler 32 fois le programme précédent.

On dispose de l'instruction :
DJNZ d codée 16, d

Elle s'applique au registre B uniquement qui, rappellons-le, ne peut contenir que la valeur maximale 255. Elle consiste à décrémenter le

contenu de ce registre, à tester s'il est nul après cette opération et, si ce n'est pas le cas, à effectuer un saut relatif ; la longueur du saut (appellé déplacement) est représentée ici par d.



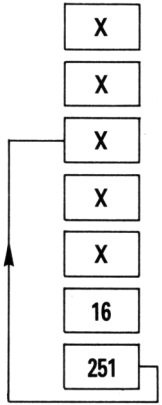
Nous savons maintenant que le programme consiste en une succession de codes en mémoire. Pour la machine, un saut **relatif** consiste à sauter vers un autre code situé à l'avant ou à l'arrière. Le nombre de codes à sauter est à préciser après DJNZ (c'est le nombre d).

Vers l'avant, il peut atteindre +127 au maximum ; vers l'arrière, le nombre d sera négatif et atteindra au plus la valeur −128.

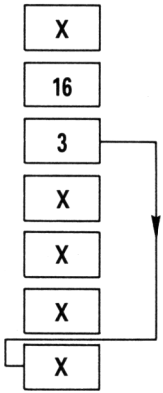
Attention : dans ce cas, il faut compter le code représentant le nombre d lui-même.
Pour représenter un nombre négatif (maxi −128), on prendra comme valeur pour d 256 − la valeur absolue (sans le signe donc) du nombre négatif.

Prenons quelques exemples :
pour −5 le code de d=256−5=251
pour −86 le code de d=256−86=17Ø

Autre représentation du **sauf relatif** :



saut arrière de −5
codé 251
DJNZ −5



saut avant de 3
codé 3
DJNZ 3

Voici notre programme, avec les adresses d'implantation, les mnémoniques et les codes :

adresses	mnémoniques	codes
16514	LD HL, (16396)	42, 12, 64
16517	LD B, 32	6, 32
16519	INC HL	35
1652Ø	LD (HL), 128	54, 128
16522	DJNZ - 5	16, 251
16524	RET	2Ø1

LD B, d se code 6, d
Dans notre cas, d = 32 car nous désirons effectuer une boucle 32 fois. Nous avions déjà vu comment entrer les codes en mémoire à l'aide de POKE à partir de l'adresse 16514.
Pensez cette fois-ci à réserver au moins 11 emplacements en mémoire à l'aide de REM.

Voici ce que vous devez obtenir :

```
 1 REM E£RND▗▛470▜( CLS TAN
10 RAND USR 16514
```

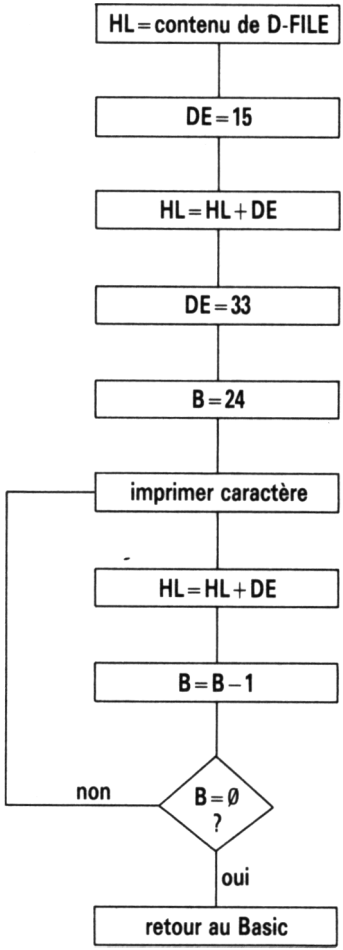
Exemple 3

Pour afficher une ligne verticale de caractères quelconques au milieu de l'écran, nous allons ajouter 15 à HL au lieu de 1 comme nous venons de le faire avec INC HL.

Il n'est toutefois pas possible d'ajouter la valeur 15 directement à la paire des registres HL ; il va falloir passer par l'intermédiaire d'une autre paire de registres, par exemple DE.

Nous aurons donc LD DE, 15 qui met la valeur 15 dans DE, mais, notez-le bien, sur 2 octets cette fois-ci.
LD DE, 15 se code 17, 15, Ø
(15 = OMS et Ø = OPS soit Ø. 256)
Il est ensuite aisé d'additionner les registres HL et DE avec
ADD HL, DE codé 25
Pour imprimer sur une ligne inférieure et dans la **même** colonne, il faut ajouter à HL la valeur 33, car n'oublions pas qu'une ligne est composée par 32 caractères + le caractère 118 = N/L.
Les instructions suivantes réalisent cette opération :
LD DE, 33 codé 17, 33,Ø
ADD HL, DE codé 25
Pour afficher sur les 24 lignes à notre disposition, nous allons utiliser une boucle répétée 24 fois à l'aide de DJNZ vu dans l'exemple 2. Le registre B contient donc la valeur 24 au départ.

Voici tout d'abord notre organigramme :



Puis le programme correspondant, à introduire toujours selon la même méthode après avoir réservé... (non, à vous de les compter) des emplacements suffisants avec 1 REM XXX...

adresses		mnémoniques	codes
16514		LD HL, (16396)	42, 12, 64
16517		LD DE, 15	17, 15, Ø
1652Ø		ADD HL, DE	25
16521		LD DE, 33	17, 33, Ø
16524		LD B, 24	6, 24
16526		LD (HL), 21	54, 21
16528		ADD HL, DE	25
16529	B ≠ 0	DJNZ - 5	16, 251
16531		RET	2Ø1

Le code 21 correspond dans notre cas au caractère à afficher, soit +.

Pour vous faire apprécier vraiment la rapidité de l'affichage en langage machine, nous vous invitons à compléter le sous-programme de l'exemple 3 avec ces quelques lignes de Basic.

Nous implantons à l'aide de POKE à chaque fois le code d'un nouveau caractère (de 1 à 63).

```
   1 REM E£RND)? ;)5 ▗/QZ;( CLS
TAN
  10 FOR I=1 TO 63
  15 POKE 16527,I
  20 RAND USR 16514
  30 NEXT I
```

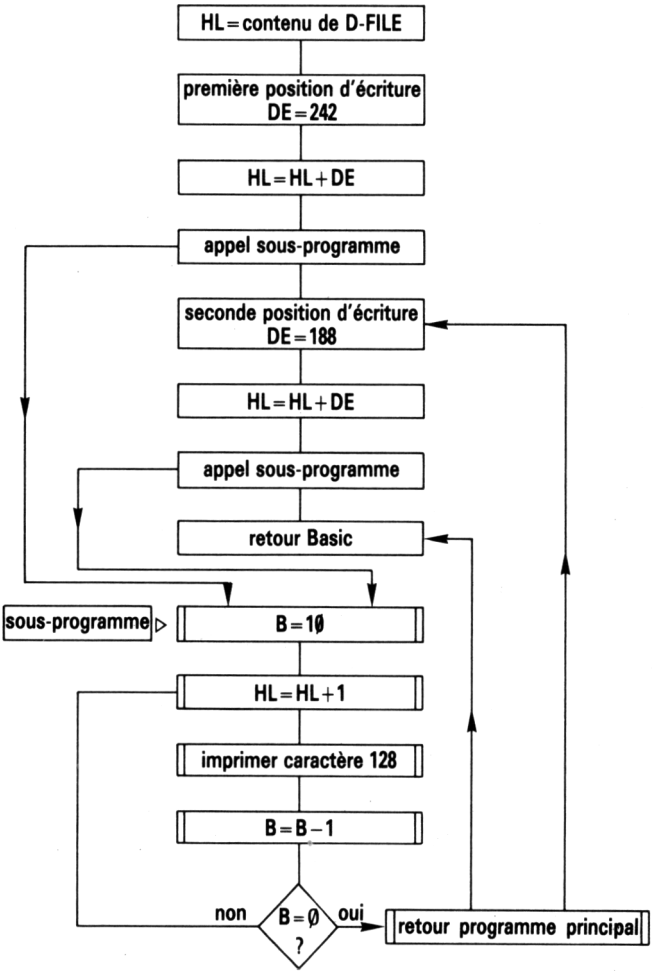
Exemple 4

A présent, il ne devrait pas être très difficile de faire apparaître sur l'écran deux lignes horizontales de même longueur, l'une sous l'autre, un peu à la manière d'un gigantesque signe égal (=).

Comme en Basic, il est astucieux et rapide de faire appel à un sous-programme, qui en l'occurence sera appelé 2 fois. Puisqu'il n'existe pas de numéros de ligne, il faudra donner une « étiquette » ou label à une portion de notre programme afin de pouvoir effectuer un branchement à cet endroit précis.

En réalité, les numéros de ligne sont ici remplacés par des adresses en mémoire.

Voici notre organigramme :



Et le programme correspondant en langage d'assemblage :

adresses	mnémoniques	codes
16514	LD HL, (16396)	42, 12, 64
16517	LD DE, 242	17, 242, Ø
1652Ø	ADD HL, DE	25
16521	CALL 16532	2Ø5, 148, 64
16524	LD DE, 188	17, 188, Ø
16527	ADD HL, DE	25
16528	CALL 16532	2Ø5, 148, 64
16531	RET	2Ø1
sous-programme		
16532	LD B, 1Ø	6, 1Ø
16534	INC HL	35
16535	LD (HL), 128	54, 128
16537	DJNZ - 5	16, 251
16539	RET	2Ø1

A noter que l'appel du sous-programme se fait par :
CALL, nn codé 2Ø5, OMS, OPS de nn

Dans notre exemple, le sous-programme débute à l'adresse 16532 qui se code, comme nous l'avons déjà vu, ainsi :
$16532 = 148 + (256 \times 64)$

Autre particularité : l'instruction RET code 2Ø1 signifie retour au programme principal si elle émane de CALL, sinon elle provoque le retour au Basic (voir aide-mémoire).

Après avoir réservé la place suffisante en mémoire à l'aide de REM XXXXX..., il convient de « POKEr » dans l'ordre les différents codes à partir de l'adresse 16514 jusqu'à l'adresse 16539, puis faites RAND USR 16514.

Exemple 5

Nous allons à présent remplir une portion de l'écran avec un caractère quelconque.

Voici ce programme dont les lignes nous sont familières déjà :

16514	LD HL, (16396)	42, 12, 64
16517	LD B, 255	6, 255
16519	→INC HL	35
16520	LD (HL), 128	54, 128
16522	DJNZ - 5	16, 251
16524	RET	201

Comme dans l'exemple 2, nous avons chargé dans le registre B la valeur maximale qu'il peut contenir soit 255.

Malheureusement, à l'exécution la machine se bloque... Si vous avez bien assimilé le chapitre consacré à l'affichage, vous connaissez sans doute déjà la raison de ce blocage.

BRAVO (juste pour ceux qui ont trouvé). Nous avons en effet modifié les caractères 118 du fichier d'affichage avec notre caractère 128 (à chaque 33e caractère).

Pour remédier à cet inconvénient, il suffira de **comparer** avec 118 le contenu de HL avant d'afficher notre propre caractère et, si la comparaison est vraie, de placer le caractère à afficher à la position suivante.

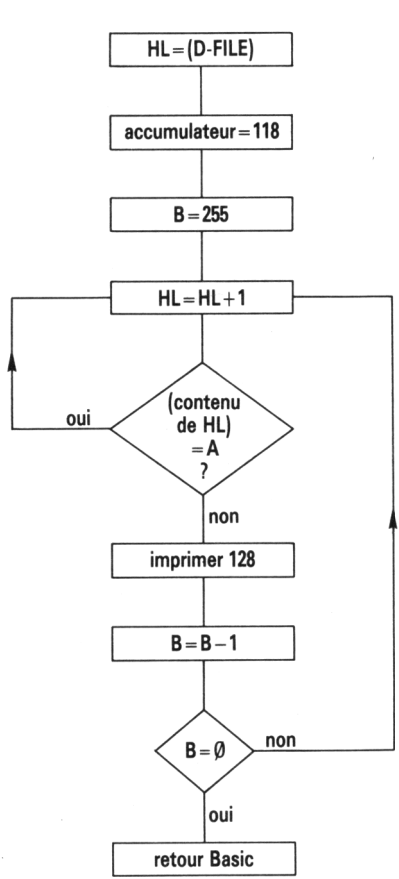
L'organigramme ci-contre vous aidera à mieux comprendre.

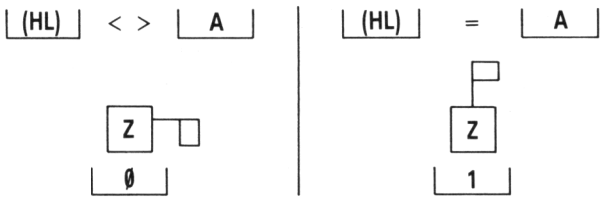
La comparaison nécessite plusieurs instructions nouvelles dont l'utilisation de l'accumulateur ou registre A.

Pour comparer 2 valeurs il est obligatoire d'en mettre une dans A et l'autre dans l'instruction de comparaison.

Ici, nous voulons comparer le contenu de HL à 118 ; chargeons tout d'abord 118 dans A avec :
LD A, 118 codé 62, 118
(attention, A ne contient qu'un octet).

Il existe l'instruction CP (HL) codée 190 qui compare le contenu de HL avec A et met à 0 le drapeau Z si les 2 valeurs sont différentes ; elle le mettra à 1 si le contenu de HL est égal à A comme l'illustre le petit schéma de la page ci-contre.


HL=(D-FILE)
accumulateur=118
B=255
HL=HL+1
(contenu de HL) =A ?
oui
non
imprimer 128
B=B−1
B=Ø
non
oui
retour Basic


(HL) < > A
Z
Ø
(HL) = A
Z
1

Rappellons que le drapeau Z est l'un des indicateurs binaires du registre F. Il ne peut prendre que 2 valeurs, Ø ou 1.

L'instruction JRZd (codée 4Ø, d) opère un branchement relatif si Z est à 1. Si Z=Ø ce saut n'est pas effectué, c'est-à-dire que cette instruction est ignorée.

Voici notre programme :

16514	LD HL, (16396)	42, 12, 64
16517	LD A, 118	62, 118
16519	LD B, 255	6, 255
16521	INC HL	35
16522	CP (HL)	19Ø
16523	JRZ - 4	4Ø, 252
16525	LD (HL), 128	54, 128
16527	DJNZ - 8	16, 248
16529	RET	2Ø1

A l'aide du petit programme Basic suivant, il vous sera très facile de rentrer tous les codes en mémoire, puis de lancer le programme.

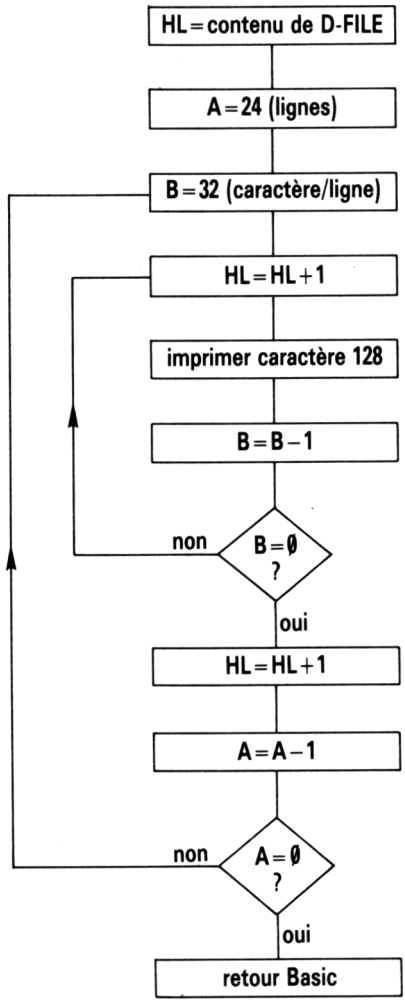
```
1 REM XXXXXXXXXXXXXXXX
2 FOR I=16514 TO 16529
4 INPUT A
6 PRINT I,A
8 POKE I,A
9 NEXT I
```

Exemple 6

Pour remplir cette fois vraiment tout l'écran avec un caractère quelconque (ici le carré noir=128), il est clair qu'il suffit de remplir 24 lignes identiques à l'aide d'une simple boucle.

A signaler que l'utilisation du caractère Ø=SPACE permet d'obtenir un effacement total et rapide comme CLS en Basic. Cela laisse également entrevoir la possibilité de n'effacer qu'une portion plus ou moins grande de l'écran.

Voyons tout d'abord l'organigramme :



Le programme en assembleur :

16514	LD HL, (16396)	42, 12, 64
16517	LD A, 24	62, 24
16519	LD B, 32	6, 32
16521	INC HL	35
16522	LD (HL), 128	54, 128
16524	DJNZ - 5	16, 251
16526	INC HL	35
16527	DEC A	61
16528	LD B, A	71
16529	DJNZ - 12	16, 244
16531	RET	201

Ce programme n'appelle aucun commentaire particulier, puisque toutes les instructions nous sont déjà presque familières.

Vous pouvez aisément le modifier, par exemple, en faisant varier le code du caractère à afficher à l'adresse 16523 avec POKE.

Voici un petit programme Basic qui vous y aidera :

```
10 REM ESSAI
20 FOR N=1 TO 63
30 POKE 16523,N
40 RAND USR 16514
50 NEXT N
```

Cet autre programme nous permettra de remplir seulement une portion de l'écran avec un caractère quelconque. Mettons pour notre exemple un carré de 100 caractères au centre de l'écran. L'organigramme est quasiment identique à celui de l'exemple précédent, sauf en ce qui concerne le nombre de lignes (10 au lieu de 24) et le nombre de colonnes (10 au lieu de 32).

La position de départ est également différente = HL + 242.

Voici le programme :

16514	LD HL, (16396)	42, 12, 64
16517	LD DE, 242	17, 242, Ø
1652Ø	ADD HL, DE	25
16521	LD A, 1Ø	62, 1Ø
16523	LD B, 1Ø	6, 1Ø
16525	INC HL	35
16526	LD (HL), 128	54, 128
16528	DJNZ - 5	16, 251
1653Ø	DEC A	61
16531	LD B, A	71
16532	LD DE, 23	17, 23, Ø
16535	ADD HL, DE	25
16536	DJNZ - 15	16, 241
16538	RET	2Ø1

Variantes :

pour modifier le nombre de lignes	POKE	16522, n
pour modifier le nombre de colonnes	POKE	16524, n
pour modifier le caractère à imprimer	POKE	16527, code (voir manuel).

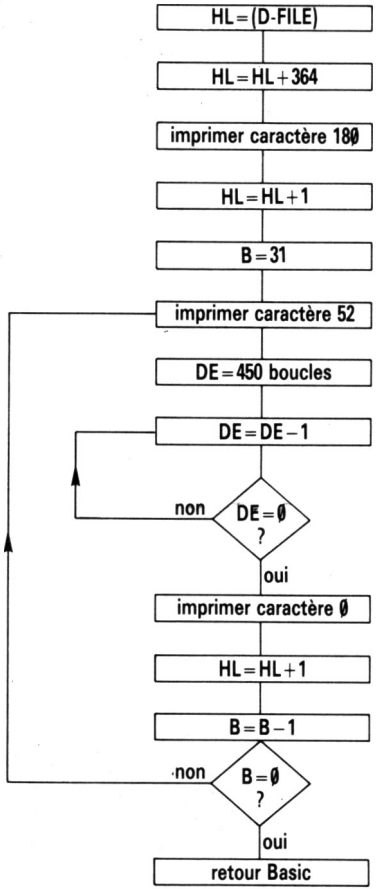
Attention ! L'octet 16533 est à modifier selon la position à obtenir pour une nouvelle ligne.

Exemple 7

Un peu d'animation à présent. Nous allons tenter de simuler le déplacement rapide d'un objet sur l'écran, plus précisément à partir de la position (D-FILE) + 364.

Pour déplacer un caractère sur l'écran, il suffit de l'imprimer, puis de l'effacer immédiatement, dans des cases successives. Toutefois, la vitesse d'exécution du langage machine est telle qu'elle ne permettrait pas de voir la moindre chose ! Nous serons donc contraints d'introduire une temporisation avant l'effacement. Ceci revient à « occuper » la machine et à lui faire exécuter plusieurs fois une boucle.

Voyons notre organigramme :



Dans ce programme, la difficulté nouvelle consiste donc à retarder l'effacement en introduisant une boucle. Or, pour espérer ralentir suffisamment notre projectile, il sera indispensable de l'effectuer un grand nombre de fois, en tout cas supérieur à 255. Nous allons donc devoir nous servir de la paire de registres DE.

Chaque nouvelle boucle décrémentera DE ; il suffira de tester si DE est nul pour passer à l'instruction suivante qui consiste à effacer le projectile.

Remarque : pour comprendre comment nous allons effectuer cette opération en langage machine, il sera utile de rappeler en quoi consiste la fonction OR :

Ø	OR	Ø=Ø
Ø	OR	1=1
1	OR	Ø=1
1	OR	1=1

En réalité, le langage d'assemblage permet de réaliser cette fonction logique entre l'accumulateur A et un registre ou une donnée. Elle effectue l'instruction OR entre ces 2 octets, et bit à bit. Le résultat sera à nouveau rangé dans A.

Exemple :

A =	1	1	1	1	Ø	Ø	1	1
E =	Ø	1	1	Ø	Ø	Ø	Ø	Ø

Après OR E A = 1 1 1 1 Ø Ø 1 1

Cette fonction agit également sur les drapeaux du registre F et plus particulièrement sur Z.

Z sera positionné à 1
si A=Ø ou plutôt ØØØØØØØØ
Z sera positionné à Ø
si A est différent de Ø
(c'est-à-dire contient au moins un bit à 1).

Revenons au test de DE ; pour savoir s'il est nul, il suffira de :

— charger le registre D dans A

LD A, D codé 122

— effectuer la fonction

OR E code 179

A sera nul si, et seulement si D et E sont nuls.

Ceci positionnera le drapeau Z à 1 si DE est nul (à Ø s'il n'est pas nul).

Ensuite il existe une instruction de **saut relatif conditionnel.**

JR NZ, d code 32, d

Comme nous l'avons déjà vu, elle effectuera le saut du nombre d'octets indiqué par d, vers l'avant ou vers l'arrière, mais cette fois-ci si Z=Ø (si Z=1 cette instruction est ignorée).

Ne mélangez pas JR Z, d qui effectue le branchement si Z=1 (exemple 4) et JR NZ, d qui l'effectue si Z=Ø.

Voici comme à l'habitude notre programme :

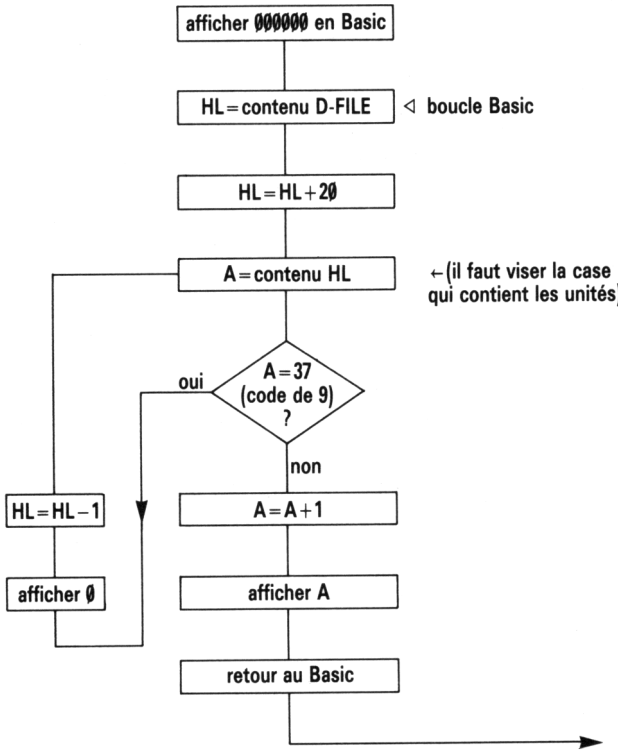
16514	LD HL, (16396)	42, 12, 64
16517	LD DE, 364	17, 1Ø8, 1
1652Ø	ADD HL, DE	25
16521	LD (HL), 18Ø	54, 18Ø
16523	INC HL	35
16524	LD B, 31	6, 31
16526	LD (HL), 52	54, 52
16528	LD DE, 45Ø	17, 194, 1
16531	DEC DE	27
16532	LD A, D	122
16533	OR E	179
16534	JR NZ - 5	32, 251
16536	LD (HL), Ø	54, Ø
16538	INC HL	35
16539	DJNZ - 15	16, 241
16541	RET	2Ø1

Utilisez le programme de chargement précédent en veillant à réserver plus de place en mémoire à l'aide de REM (la dernière adresse sera 16541).

Exemple 8

Il est souvent nécessaire de réaliser pour certains jeux un ou plusieurs compteurs très rapides, car un comptage trop lent ralentirait le jeu lui-même. Nous allons le faire à l'aide de quelques instructions que nous connaissons déjà. A signaler que ce compteur accepte un nombre quelconque de chiffres et qu'il permet d'atteindre une vitesse de comptage fantastique.

Organigramme



Le programme correspondant à l'organigramme précédent n'incrémente que d'**une** unité notre compteur ; il faudra donc l'appeler aussi souvent que nécessaire en tant que sous-programme, soit dans un autre programme en langage machine, soit encore à partir du Basic.

La seule nouvelle instruction est celle du **saut relatif inconditionnel.**

JR d codé 24, d

16514	LD HL, (16396)	42, 12, 64
16517	LD DE, 20	17, 20, 0
16520	ADD HL, DE	25
16521	LD A, (HL)	126
16522	CP 37	254, 37
16524	JR Z 3	40, 3
16526	INC A	60
16527	LD (HL), A	119
16528	RET	201
16529	LD (HL), 28	54, 28
16531	DEC HL	43
16532	JR - 13	24, 243

Vous introduirez les codes selon la méthode habituelle, puis à l'aide du programme Basic suivant vous pourrez voir défiler les chiffres.

Vous aurez bien noté qu'il faut que les compteurs soient mis à 0 sur l'écran avec le lancement du programme machine (ligne 10).

```
 1 REM EERND) = ;?TAN 00F/ NEXT

10 PRINT TAB 14;"000000"
20 RAND USR 16514
30 GOTO 20
```

Exemple 9

En langage machine également, bien souvent, nous serons amenés à utiliser le clavier pour donner un ordre ou agir directement sur le déroulement du programme.

Nous allons apprendre comment, en langage machine, il est possible de tester si une ou plusieurs touches ont été sollicitées ; il nous appartiendra de définir au préalable quelles touches doivent être lues par la machine.

Voici le sous-programme qui réalise cette fonction et permet de tester **une seule touche à la fois.**

LD C, 254	codes 14, 254	
LD B, valeur selon touche	code 6, d	(voir tableau
IN A, (C)	codes 237, 12Ø	de codage)
AND valeur selon tableau	code 23Ø, d	

Si la touche codée a été enfoncée, le drapeau Z sera mis à 1 ; il suffira ensuite d'utiliser Z dans une instruction de saut conditionnel (JR Z ou encore JR NZ).

Pour sélectionner plusieurs touches dans un même programme, il faudra écrire ces lignes autant de fois que nécessaire.

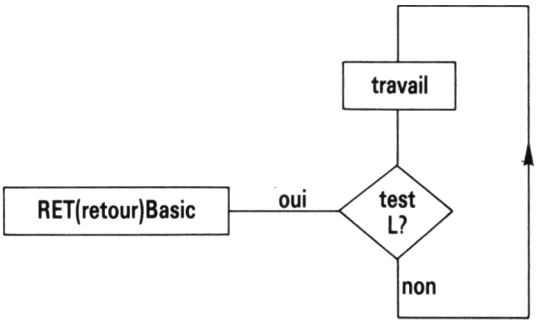
Attention : chaque touche est codée différemment selon le tableau suivant :

Tableau de codage des touches

	AND 1	**AND 2**	**AND 4**	**AND 8**	**AND 16**
LD B, 254	SHIFT	Z	X	C	V
LD B, 253	A	S	D	F	G
LD B, 251	Q	W	E	R	T
LD B, 247	1	2	3	4	5
LD B, 239	Ø	9	8	7	6
LD B, 223	P	O	I	U	Y
LD B, 191	N/L	L	K	J	H
LD B, 127	SPACE	•	M	N	B

L'exemple suivant fera défiler (très vite) les 64 premiers caractères à la première position d'affichage. Nous choisirons la touche L pour stopper le déroulement du programme et revenir au Basic. Sans cette précaution, le programme en langage machine tournerait indéfiniment (boucle fermée).

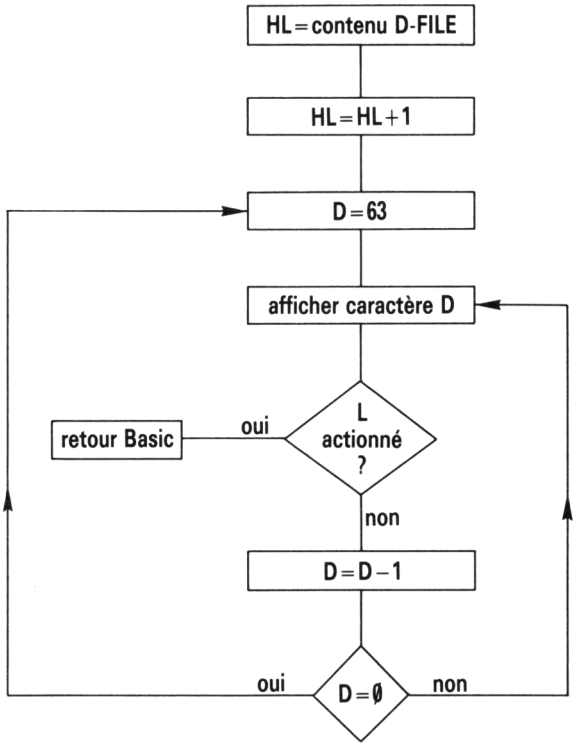
Organigramme



Programme

16514	LD HL, (16396)	42, 12, 64
16517	INC HL	35
16518	LD D, 63	22, 63
1652Ø	LD (HL), D	114
16521	LD C, 254	14, 254
16523	LD B, 191	6, 191
16525	IN A, (C)	237, 12Ø
16527	AND 2	23Ø, 2
16529	RET Z	2ØØ
1653Ø	NOP	Ø
16531	DEC D	21
16532	LD A, D	122
16533	CP Ø	254, Ø
16535	JR NZ - 17	32, 239
16537	JR - 21	24, 235

NOP ne fait rien ! Il évite ici un effet stroboscopique sur l'écran. (Voir aide-mémoire).



DEUXIÈME PARTIE

programmes

ZIG-ZAG

Nous vous offrons à présent une petite récréation à l'aide d'un programme très court, mais ô combien captivant. Il met en œuvre bon nombre des instructions déjà étudiées et en particulier 2 tests de touche.

Nous l'avons intitulé ZIG-ZAG et vous comprendrez très vite pourquoi !

Pour vous faciliter davantage encore l'introduction des codes en mémoire, le programme comporte 3 parties, à savoir réservation de la place (ligne 1), présentation **alignée** des divers codes (à respecter scrupuleusement avec tous les espaces) puis quelques lignes qui chargent automatiquement les codes en mémoire.

Nous insistons sur la nécessité de sauver le programme avant son lancement par RUN 900, car la moindre erreur sur les codes provoquera probablement un blocage de la machine. Tapez ces lignes.

```
   1 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  10 LET A$="[redacted]"
  20 PRINT AT 5,1;A$;AT 13,1;A$
  30 PRINT AT 9,16;A$;AT 17,16;A
$
  40 PRINT AT 20,1;A$
  50 PRINT AT 21,0;"------------
--------------------"
  60 RAND USR 16514
  70 PRINT AT 21,10;"[redacted]"
  80 FOR I=1 TO 200
  81 NEXT I
  85 CLS
  90 GOTO 10
 900 LET A$=""
 910 FAST
1000 LET A$=A$+"042 012 064 017
016 000 025 054 052 017 208 007
027 122 179 032 251 054 000 017
033 000 025 126 254 148 040 228
"
```

```
1010 LET A$=A$+"254 022 200 001
254 253 237 120 230 001 032 001
043 001 254 191 237 120 230 001
032 001 035 054 052 024 210 "
2200 LET A=1
2210 FOR I=1 TO 55
2220 POKE 16513+I,VAL A$(A TO A+
2)
2230 LET A=A+4
2240 NEXT I
2250 SLOW
2260 REM SAUVEZ LE PROGRAMME
AVANT DE FAIRE RUN 900
3000 RUN
```

En lisant ces lignes, vous avez certainement déjà sur l'écran le décor du jeu, qui consiste à faire zigzaguer la balle entre les obstacles et lui faire atteindre le bas.

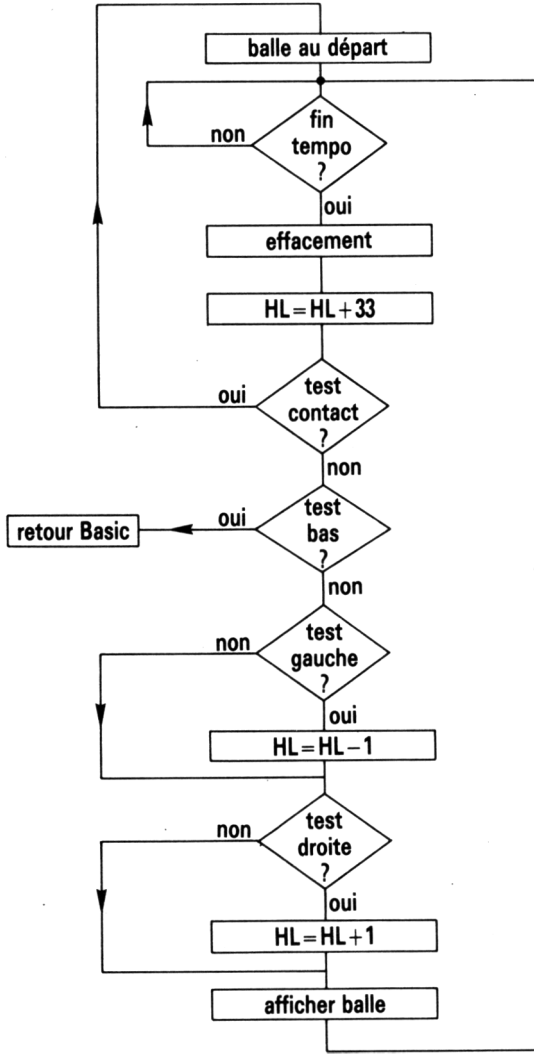
Pour ce faire, vous disposez des touches A et N/L qui dévient respectivement la balle vers la gauche et vers la droite.

Le parcours est difficile mais, pour parvenir à vos fins, sachez que les obstacles peuvent être « grignotés » en les attaquant de biais et par leur extrémité.

Pour les virtuoses seulement, l'intervention sur l'octet à l'adresse 16525 (poids fort de la temporisation), permettra d'atteindre des vitesses époustouflantes.

Faites POKE 16525, d (d variant de Ø à 6).

Nous vous donnons maintenant l'organigramme complet du Zigzag.


balle au départ
fin tempo ?
non
oui
effacement
HL=HL+33
test contact ?
oui
non
test bas ?
oui
retour Basic
non
test gauche ?
non
oui
HL=HL−1
test droite ?
non
oui
HL=HL+1
afficher balle

A ce stade de l'ouvrage, une mise au point est nécessaire : le travail le plus fastidieux en langage machine consiste à retrouver les codes correspondant aux mnémoniques et surtout à calculer la valeur des sauts relatifs et la décomposition sur 2 octets des nombres sur 16 bits.

Il existe un outil très commode qui réalise cette tâche à notre place : **l'assembleur.**

C'est un logiciel sophistiqué qui permet d'écrire les instructions en mnémotique d'une manière très souple. Son grand avantage est qu'il accepte l'emploi « d'étiquettes », un peu comme les numéros de ligne en Basic.
Le logiciel se charge à notre place de tous les calculs de saut ; en effet, il suffit de lui indiquer l'étiquette du saut à effectuer (l'étiquette étant considérée ici par la machine comme une adresse).

Il n'est pas raisonnable ou du moins très difficile de concevoir de longs programmes sans l'aide d'un tel assembleur, car le risque d'erreur s'accroît avec leur longueur. Rappelez-vous qu'une seule d'entre elles suffira à bloquer la machine.

Pour notre dernier jeu, ZIG-ZAG, et tous les programmes suivants, nous avons utilisé l'assembleur ARTIC, vendu à un prix dérisoire comparativement au travail qu'il nous épargne (environ 75,00 F avec son manuel détaillé).

Voici à nouveau le programme de ZIG-ZAG tel qu'il se présente sur l'écran.
Sur cet assembleur, les nombres décimaux sont toujours précédés par + ou −, sinon ils sont considérés comme des nombres hexadécimaux. Le nom des étiquettes est toujours choisi par le programmeur.

```
DEBUT LD HL,(+16396)
      LD DE,+16
      ADD HL,DE
      LD (HL),+52
TEMPO LD DE,+2000
LAB0  DEC DE
      LD A,D
      OR E
      JR NZ LAB0
      LD (HL),0
      LD DE,+33
      ADD HL,DE
      LD A,(HL)
      CP +148
```

```
        JR Z DEBUT
        CP+22
        RET Z
        LD C,+254
        LD B,+253
        IN A,(C)
        AND 1
        JR NZ SUIT1
        DEC HL
SUIT1   LD C,+254
        LD B,+191
        IN A,(C)
        AND 1
        JR NZ SUIT2
        INC HL
SUIT2   LD (HL),+52
        JR TEMPO
```

Enfin, voici une autre présentation des premières lignes du programme précédent :

adresses	mnémoniques	codes
DEBUT = 16514	LD HL, (+16396)	42, 12, 64
	LD DE, +16	17, 16, Ø
	ADD HL, DE	25
	LD (HL), +52	54, 52
TEMPO = 16523	LD DE, +2ØØØ	17, 2Ø8, 7
LABØ = 16526	DEC D	27
	LD A, D	122
	OR E	179
	JR NZ LABØ	32, 251

Pour aller à l'étiquette LABØ, il est bien nécessaire de faire un saut en arrière de −5 codé 251.

Exemple 10

Nous ne disposons que de 3 paires de registres. Une question se pose : comment pouvoir traiter et mémoriser plus de 3 données ? La réponse est fort simple. Il suffit de réserver à la fin du pro-

gramme, à des adresses précises donc connues, assez d'octets pour y stocker nos données. Si ces dernières sont sur 16 bits, il faudra réserver 2 octets consécutifs.

Voici un exemple :

16514	———	programme
16515	———	
•	•	
•	•	
•	•	
•	•	
•	•	
18499	———	
18500	0	2 octets pour un nombre sur 16 bits
18501	0	
18502	0	
18503	0	
18504	0	
18505	0	

Chaque fois que l'on voudra garder la valeur de HL, de BC, ou de DE, il suffira de faire LD (18500), HL ou LD (18502), BC ou LD (18504), DE. HL, BC, DE pourront dès lors être utilisés pour traiter d'autres données. Pour récupérer les données stockées à ces adresses, on fera LD HL,(18500) ou LD BC,(18502) ou LD DE,(18504). Vous ne serez plus étonné de voir que les derniers codes de nos longs programmes (dans lesquels beaucoup de données sont à traiter), sont tous à 0.

Ici également, l'utilisation d'un assembleur nous rendra la tâche plus facile grâce à l'emploi d'étiquettes de notre choix à la place des adresses. Lors de l'assemblage, l'assembleur saura à quelle adresse correspond chaque étiquette (cf. langage d'assemblage du chronomètre).

La spirale infernale

Avant de vous dévoiler la partie récréative de notre ouvrage, nous souhaitons vous faire travailler un peu à votre tour. Plus précisément, nous vous invitons à coder vous-même (à titre de révision) le petit programme suivant, à l'aide des instructions utilisées dans les divers exemples ou encore en consultant l'aide-mémoire en fin de recueil.

Pour faciliter votre tâche, nous vous présentons l'organigramme et le programme complet en langage d'assemblage, ainsi qu'un petit programme Basic complémentaire.

But du jeu : il s'agit pour le joueur de parvenir à atteindre le centre d'une spirale sans jamais heurter les parois entre lesquelles vous vous déplacez. Dans le cas contraire, vous vous retrouverez irrémédiablement au départ.

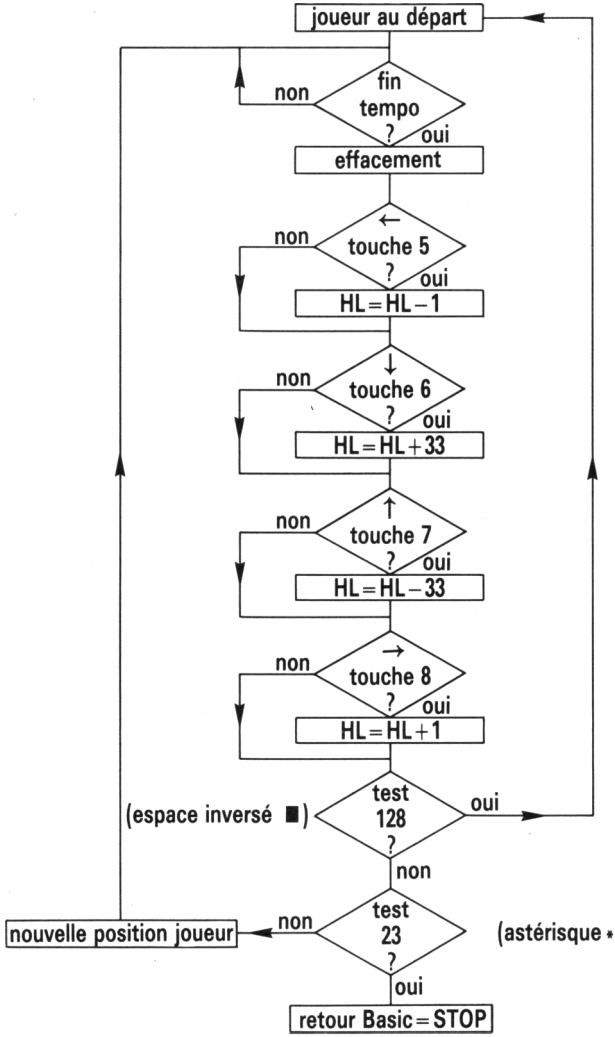
Cela exige une bonne maîtrise de soi ; les 4 touches fléchées du clavier vous aideront dans vos déplacements.

Remarque : la solution de cet exercice se trouve en fin d'ouvrage mais, bien entendu, il ne sera nécessaire de la consulter qu'à titre de contrôle !

Programme Basic

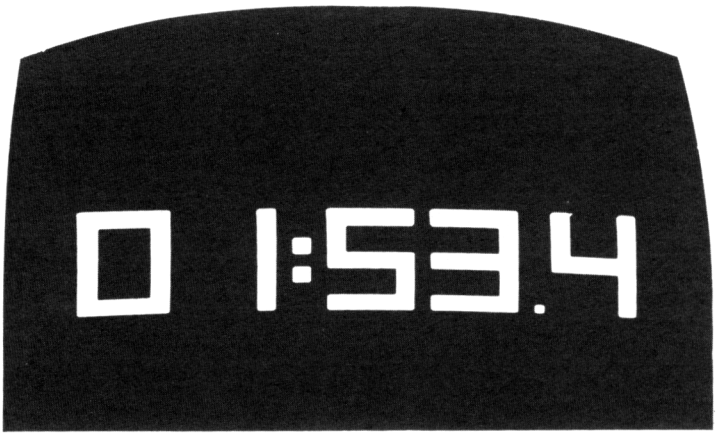
```
  8 LET X=0
  9 LET Y=-1
 10 LET N=1
 20 LET G=22
 25 FAST
 30 FOR I=1 TO G
 40 LET X=X+COS (N*PI/2)
 50 LET Y=Y+SIN (N*PI/2)
 60 PRINT AT X,Y;"■"
 70 NEXT I
 80 LET G=G-1
 90 LET N=N+1
 95 IF N=23 THEN GOTO 200
100 GOTO 25
200 SLOW
220 PRINT AT 11,10;"*"
250 RAND USR 16514
300 PRINT AT 0,0;"    BRAVO
999 STOP
```

```
DEBUT LD HL,(+16396)
      LD DE,+34
      ADD HL,DE
      LD(HL),+52
TEMPO LD DE,+2000
ETIQ1 DEC DE
      LD A,D
      OR E
      JR NZ ETIQ1
      LD(HL),0
      LD C,+254
      LD B,+247
      IN A,(C)
      AND +16
      JR NZ ETIQ2
      DEC HL
ETIQ2 LD C,+254
      LD B,+239
      IN A,(C)
      AND +16
      JR NZ ETIQ3
      LD DE,+33
      ADD HL,DE
ETIQ3 LD C,+254
      LD B,+239
      IN A,(C)
      AND 8
      JR NZ ETIQ4
      LD DE,-33
      ADD HL,DE
ETIQ4 LD C,+254
      LD B,+239
      IN A,(C)
      AND 4
      JR NZ ETIQ5
      INC HL
ETIQ5 LD A,(HL)
      CP +128
      JR Z DEBUT
      LD A,(HL)
      CP +23
      RET Z
      LD(HL),+52
      JP TEMPO
```


joueur au départ
fin tempo ?
non
oui
effacement
← touche 5 ?
non
oui
HL=HL−1
↓ touche 6 ?
non
oui
HL=HL+33
↑ touche 7 ?
non
oui
HL=HL−33
→ touche 8 ?
non
oui
HL=HL+1
test 128 ?
(espace inversé ■)
oui
non
test 23 ?
non
(astérisque *)
nouvelle position joueur
oui
retour Basic=STOP

CHRONOMÈTRE DIGITAL

Géant



Le chronomètre digital que nous proposons est une réalisation 100 % langage machine ; l'affichage est constitué par des chiffres géants à 7 segments. Nous avons choisi ce sujet pour sa construction méthodique, la possibilité d'utiliser de nombreuses instructions simples et surtout pour bien mettre en valeur la prodigieuse rapidité du langage machine ; en effet, nous affichons le 1/10 de seconde.

Essayez donc en Basic pour voir !...

Notre chronomètre dispose des commandes suivantes :

START	touche S
STOP	touche A
LAP (temps intermédiaire)	touche L à maintenir
RESET (remise à Ø après STOP)	touche Ø

Une action sur la touche 9 permet de simuler BREAK pour un retour au Basic.

La base de temps du chrono utilise l'octet de poids faible de la variable système FRAMES à l'adresse 16436.

Afin de bien donner à cet exemple une valeur pédagogique, nous vous proposons de photocopier les pages regroupant le programme en langage d'assemblage et de les réunir dans l'ordre pour étudier la construction du programme.

Ce faisant, il vous sera très aisé de retrouver les étiquettes des divers sauts et, qui sait, de retrouver également tous les codes contenus dans la grande variable A$ du programme de lancement.

Nous souhaitons que l'expérience acquise au cours de cette étude vous soit profitable et contribue à vous faire apprécier le langage machine à sa juste valeur malgré ses sévères exigences.

Remarque : pour réserver une REM de grande dimension, il n'est pas pratique d'aller au-delà de 3 ou 4 lignes de caractères quelconques à la fois. Il est également plus agréable de remplir des lignes entières pour simplifier le comptage des caractères. Ensuite, il est très aisé de dupliquer plusieurs fois la première REM à l'aide de la touche EDIT. Il reste un problème : comment faire admettre à la machine que toutes les lignes REM n'en forment qu'une seule pour la réservation ? Il suffit de mettre dans les octets 16511 et 16512 la longueur totale.

Pour nous POKE 16511, 135 et POKE 16512, 2
soit $135 + (256 \times 2) = 647$ octets réservés

En ayant recours à de nombreux sous-programmes, il a été possible de structurer au maximum le programme en langage d'assemblage du chronomètre.

L'instruction CALL nn est utilisée à cet effet. (Voir aide-mémoire à la fin de l'ouvrage).

Programme complet Basic

```
    1 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    2 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    3 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    4 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    5 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    6 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    7 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   80 POKE 16511,135
   90 POKE 16512,2
   95 REM RIEN ENTRE LES ""
  100 LET A$=""
  200 FAST
```

```
1000 LET A$=A$+"000 000 062 000
050 206 066 050 207 066 050 208
066 050 209 066 050 210 066 042
012 064 017 009 001 025 034 219
"

1010 LET A$=A$+"066 017 006 000
025 034 217 066 017 008 000 025
034 215 066 017 006 000 025 034
213 066 017 007 000 025 034 211
"

1020 LET A$=A$+"066 042 012 064
017 087 001 025 054 128 017 066
000 025 054 128 017 080 000 025
054 004 205 096 065 001 254 239
"

1030 LET A$=A$+"237 120 230 001
040 168 237 120 230 002 200 001
254 253 237 120 230 002 032 233
062 005 050 052 064 001 254 191
"

1040 LET A$=A$+"237 120 230 002
196 096 065 058 052 064 254 000
032 249 062 005 050 052 064 001
254 253 237 120 230 001 040 197
"

1050 LET A$=A$+"058 206 066 060
050 206 066 254 010 032 214 062
000 050 206 066 058 207 066 060
050 207 066 254 010 032 198 062
"

1060 LET A$=A$+"000 050 207 066
058 208 066 060 050 208 066 254
006 032 182 062 000 050 208 066
058 209 066 060 050 209 066 254
"

1070 LET A$=A$+"010 032 166 062
000 050 209 066 058 210 066 060
050 210 066 254 006 032 150 062
000 050 210 066 024 143 042 211
"

1080 LET A$=A$+"066 058 206 066
205 142 065 042 213 066 058 207
066 205 142 065 042 215 066 058
208 066 205 142 065 042 217 066
"

1090 LET A$=A$+"058 209 066 205
142 065 042 219 066 058 210 066
205 142 065 201 205 085 066 254
000 032 019 205 141 066 205 160
"

1100 LET A$=A$+"066 205 163 066
205 182 066 205 190 066 205 198
066 201 254 001 032 007 205 190
066 205 198 066 201 254 002 032
"

1110 LET A$=A$+"016 205 141 066
205 190 066 205 152 066 205 182
066 205 160 066 201 254 003 032
016 205 141 066 205 152 066 205
"
```

```
1120 LET A$=A$+"160 066 205 190
066 205 198 066 201 254 004 032
013 205 168 066 205 152 066 205
190 066 205 198 066 201 254 005
"
1130 LET A$=A$+"032 016 205 141
066 205 152 066 205 160 066 205
168 066 205 198 066 201 254 006
032 019 205 141 066 205 152 066
"
1140 LET A$=A$+"205 160 066 205
168 066 205 182 066 205 198 066
201 254 007 032 010 205 141 066
205 190 066 205 198 066 201 254
"
1150 LET A$=A$+"008 032 022 205
141 066 205 152 066 205 160 066
205 168 066 205 182 066 205 190
066 205 198 066 201 254 009 192
"
1160 LET A$=A$+"205 141 066 205
152 066 205 160 066 205 168 066
205 190 066 205 198 066 201 034
221 066 205 116 066 017 004 000
"
1170 LET A$=A$+"025 205 116 066
205 130 066 017 099 000 025 205
130 066 017 198 000 025 205 130
066 201 006 007 017 033 000 054
"
1180 LET A$=A$+"000 025 016 251
042 221 066 201 006 004 054 000
035 016 251 042 221 066 201 006
005 054 128 035 016 251 042 221
"
1190 LET A$=A$+"066 201 017 099
000 025 205 141 066 201 017 198
000 025 205 141 066 201 006 004
017 033 000 054 128 025 016 251
"
1200 LET A$=A$+"042 221 066 201
017 099 000 025 205 168 066 201
017 004 000 025 205 168 066 201
017 103 000 025 205 168 066 201
"
1210 LET A$=A$+"000 000 000 000
000 000 000 000 000 000 000 000
000 000 000 000 000 000 "

2200 LET A=1
2210 FOR I=1 TO 606
2220 POKE 16513+I,VAL A$(A TO A+
2)
2230 LET A=A+4
2240 NEXT I
2250 SLOW
2270 POKE 16514,118
2280 POKE 16515,118
2290 REM SAUVEZ LE PROGRAMME
AVANT DE FAIRE RUN
2500 REM ......................
3000 RAND USR 16516
```

Note : *En photocopiant les pages suivantes et en les assemblant, vous pourrez reconstituer le programme complet en langage d'assemblage du chronomètre. Cet exercice constitue un excellent résumé des instructions de base du ZX 80.*

Programme en langage d'assemblage

PARTIE 1

```
        NOP
        NOP
        ;INITIALISATIONS
DEBUT   LD A,0
        LD (DIZ),A
        LD (SECU),A
        LD (SECD),A
        LD (MINU),A
        LD (MIND),A
        LD HL,(+16396)
        LD DE,+265
        ADD HL,DE
        LD (AFMD),HL
        LD DE,6
        ADD HL,DE
        LD (AFMU),HL
        LD DE,8
        ADD HL,DE
        LD (AFSD),HL
        LD DE,6
        ADD HL,DE
        LD (AFSU),HL
        LD DE,7
        ADD HL,DE
        LD (AFDI),HL
        LD HL,(+16396)
        LD DE,+343
        ADD HL,DE
        LD (HL),+128
        LD DE,+66
        ADD HL,DE
        LD (HL),+128
        LD DE,+80
        ADD HL,DE
        LD (HL),4

        CALL AFFIC
```

PARTIE 2

```
        ;TEST RAZ-BREAK-START
ICI1    LD BC,EFFE
        IN A,(C)
        AND 1
        JR Z DEBUT
        IN A,(C)
        AND 2
        RET Z
        LD BC,FDFE
        IN A,(C)
        AND 2
        JR NZ ICI1
        LD A,5
        LD (+16436),A

        ;TEST LAP
RECOM   LD BC,BFFE
        IN A,(C)
        AND 2
        CALL NZ AFFIC

        ;TEMPORISATION
TEMPO   LD A,(+16436)
        CP 0
        JR NZ TEMPO
        LD A,5
        LD (+16436),A

        ;TEST STOP
        LD BC,FDFE
        IN A,(C)
        AND 1
        JR Z ICI1
```

PARTIE 3

```
;INCREMENTATION
LD A,(DIZ)
INC A
LD (DIZ),A
CP +10
JR NZ RECOM
LD A,0
LD (DIZ),A
LD A,(SECU)
INC A
LD (SECU),A
CP +10
JR NZ RECOM
LD A,0
LD (SECU),A
LD A,(SECD)
INC A
LD (SECD),A
CP 6
JR NZ RECOM
LD A,0
LD (SECD),A
LD A,(MINU)
INC A
LD (MINU),A
CP +10
JR NZ RECOM
LD A,0
LD (MINU),A
LD A,(MIND)
INC A
LD (MIND),A
CP 6
JR NZ RECOM
LD A,0
LD (MIND),A
JR RECOM
```

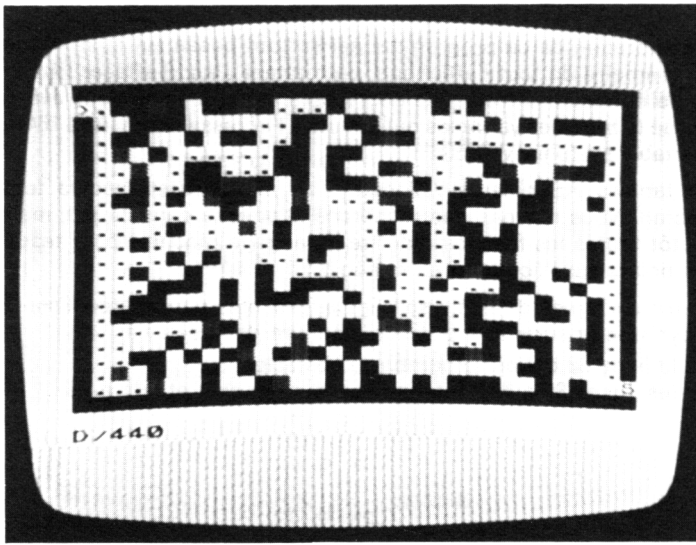
```
        ;AFFICHAGE
AFFIC   LD HL,(AFDI)
        LD A,(DIZ)
        CALL CHIFF
        LD HL,(AFSU)
        LD A,(SECU)
        CALL CHIFF
        LD HL,(AFSD)
        LD A,(SECD)
        CALL CHIFF
        LD HL,(AFMU)
        LD A,(MINU)
        CALL CHIFF
        LD HL,(AFMD)
        LD A,(MIND)
        CALL CHIFF
        RET

        ;SPRG AFFICHAGE CHIFFRE
CHIFF   CALL EFFAC
        CP 0
        JR NZ CAS1
        CALL SEG1
        CALL SEG3
        CALL SEG4
        CALL SEG5
        CALL SEG6
        CALL SEG7
        RET
CAS1    CP 1
        JR NZ CAS2
        CALL SEG6
        CALL SEG7
        RET
CAS2    CP 2
        JR NZ CAS3
        CALL SEG1
        CALL SEG6
        CALL SEG2
        CALL SEG5
        CALL SEG3
        RET
```

```
CAS3    CP 3
        JR NZ CAS4
        CALL SEG1
        CALL SEG2
        CALL SEG3
        CALL SEG6
        CALL SEG7
        RET
CAS4    CP 4
        JR NZ CAS5
        CALL SEG4
        CALL SEG2
        CALL SEG6
        CALL SEG7
        RET
CAS5    CP 5
        JR NZ CAS6
        CALL SEG1
        CALL SEG2
        CALL SEG3
        CALL SEG4
        CALL SEG7
        RET
CAS6    CP 6
        JR NZ CAS7
        CALL SEG1
        CALL SEG2
        CALL SEG3
        CALL SEG4
        CALL SEG5
        CALL SEG7
        RET
CAS7    CP 7
        JR NZ CAS8
        CALL SEG1
        CALL SEG6
        CALL SEG7
        RET
CAS8    CP 8
        JR NZ CAS9
        CALL SEG1
        CALL SEG2
        CALL SEG3
        CALL SEG4
        CALL SEG5
        CALL SEG6
        CALL SEG7
        RET
```

```
CAS9   CP 9
       RET NZ
       CALL SEG1
       CALL SEG2
       CALL SEG3
       CALL SEG4
       CALL SEG6
       CALL SEG7
       RET

       ;SPRG EFFACEMENT
EFFAC  LD (RAPP),HL
       CALL ENLV1
       LD DE,4
       ADD HL,DE
       CALL ENLV1
       CALL ENLV2
       LD DE,+99
       ADD HL,DE
       CALL ENLV2
       LD DE,+198
       ADD HL,DE
       CALL ENLV2
       RET

ENLV1  LD B,7
       LD DE,+33
LAB0   LD (HL),0
       ADD HL,DE
       DJNZ LAB0
       LD HL,(RAPP)
       RET

ENLV2  LD B,4
LAB1   LD (HL),0
       INC HL
       DJNZ LAB1
       LD HL,(RAPP)
       RET

       ;SPRG SEGMENTS
SEG1   LD B,5
LBL1   LD (HL),+128
       INC HL
       DJNZ LBL1
       LD HL,(RAPP)
       RET
```

```
SEG2   LD DE,+99
       ADD HL,DE
       CALL SEG1
       RET
SEG3   LD DE,+198
       ADD HL,DE
       CALL SEG1
       RET
SEG4   LD B,4
       LD DE,+33
LBL2   LD (HL),+128
       ADD HL,DE
       DJNZ LBL2
       LD HL,(RAPP)
       RET
SEG5   LD DE,+99
       ADD HL,DE
       CALL SEG4
       RET
SEG6   LD DE,4
       ADD HL,DE
       CALL SEG4
       RET
SEG7   LD DE,+103
       ADD HL,DE
       CALL SEG4
       RET

       ;VARIABLES
DIZ    00
SECU   00
SECD   00
MINU   00
MIND   00
AFDI   0000
AFSU   0000
AFSD   0000
AFMU   0000
AFMD   0000
RAPP   0000

FIN    NOP
```

DÉDALE



Il s'agit, comme dans tout labyrinthe d'ailleurs, de rejoindre la sortie. Le plan apparaît en début de partie pour vous laisser pendant quelques instants la possibilité d'en mémoriser les contours.

Vous n'en aurez cependant guère le temps, car vous voilà plongé dans les ténèbres et impossible de rebrousser chemin car l'issue est obturée. Il faut donc avancer à tâtons, à l'aide des touches de déplacement (←↑→↓) du clavier.

Soyez prudent, car à chaque fois que vous rencontrez un obstacle il se matérialise sur l'écran comme pour vous dissuader de prendre cette direction. Mais attention, certains obstacles ont la fâcheuse propriété de vous renvoyer au départ, selon le sens dans lequel vous les heurtez.

Heureusement, il reste une trace de votre passage vous permettant de contourner certains obstacles la seconde fois. Si vous parvenez à déjouer toutes les embûches que vous présente votre ZX 81 et à atteindre la sortie, vous aurez la satisfaction de voir se reconstituer le plan initial de votre labyrinthe qui quelquefois est plus aisé à traverser que d'autres !

• La technique

L'élaboration de ce programme est surtout basée sur le stockage en mémoire de tout l'affichage, permettant ainsi d'effacer l'écran et de continuer à tester les déplacements du joueur. Nous avons utilisé la zone des variables qui débute à l'adresse contenue à 16400 (variable système VARS).

Le tableau A$ réserve 792 octets en mémoire et recevra tout l'écran. Nous n'avons pas recherché la vitesse dans ce jeu, mais plutôt toutes les facilités qu'offre le langage machine pour tester les points sur l'écran ou en mémoire.

Quelques lignes de Basic sont indispensables pour matérialiser le labyrinthe et surtout la position aléatoire des obstacles :

— la ligne 80 définit le nombre d'obstacles,
— les lignes 85 à 120 déterminent leur position et leur type.

Programme en langage d'assemblage

```
        NOP
        NOP
DEBUT   LD HL,(+16396)
        LD DE,+35
        ADD HL,DE
        LD (BAECX),HL
        LD HL,(+16400)
        LD DE,+40
        ADD HL,DE
        LD (BAME),HL

AFFIC   LD HL,(BAECX)
        LD (HL),+52

        LD HL,(BAME)
TESTG   INC HL
        LD A,(HL)
        CP +56
        RET Z

TEMPO   LD DE,+2000
LAB0    DEC DE
        LD A,D
        OR E
        JR NZ LAB0

        LD HL,(BAECX)
EFFAC   LD (HL),+27

        LD C,+254
CAS8    LD B,+239
        IN A,(C)
        AND 4
        JR NZ CAS5
        LD HL,(BAME)
        INC HL
        LD A,(HL)
        CP 8
```

```
        JP Z SPE
        CP 0
        JR NZ SUIT1
        LD (BAME),HL
        LD HL,(BAECX)
        INC HL
        LD (BAECX),HL
        JR AFFIC
SUIT1   LD HL,(BAECX)
        INC HL
        LD (HL),+151
        JR AFFIC
CAS5    LD B,+247
        IN A,(C)
        AND +16
        JR NZ CAS7
        LD HL,(BAME)
        DEC HL
        LD A,(HL)
        CP 0
        JR NZ SUIT2
        LD (BAME),HL
        LD HL,(BAECX)
        DEC HL
        LD (BAECX),HL
        JR AFFIC
SUIT2   LD HL,(BAECX)
        DEC HL
        LD (HL),+151
        JR AFFIC
CAS7    LD B,+239
        IN A,(C)
        AND 8
        JR NZ CAS6
        LD HL,(BAME)
        LD DE,-33
        ADD HL,DE
        LD A,(HL)
        CP 0
        JR NZ SUIT3
```

```
        LD  (BAME),HL
        LD HL,(BAECX)
        ADD HL,DE
        LD  (BAECX),HL
        JP AFFIC
SUIT3   LD HL,(BAECX)
        ADD HL,DE
        LD  (HL),+151
        JP AFFIC
CAS6    LD B,+239
        IN A,(C)
        AND +16
        JR NZ CASBR
        LD HL,(BAME)
        LD DE,+33
        ADD HL,DE
        LD A,(HL)
        CP 0
        JR NZ SUIT4
        LD  (BAME),HL
        LD HL,(BAECX)
        ADD HL,DE
        LD  (BAECX),HL
        JP AFFIC
SUIT4   LD HL,(BAECX)
        ADD HL,DE
        LD  (HL),+151
        JP AFFIC
CASBR   LD B,+127
        IN A,(C)
        AND 1
        RET Z
        JP AFFIC
SPE     LD HL,(BAECX)
        INC HL
        LD  (HL),8
        JP DEBUT
BAECX   0000
BAME    0000
```

Programme complet Basic

```
    1 REM XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    2 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    3 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   10 DIM A$(792)
   12 FAST
   15 LET U$="▇"
   20 FOR Y=0 TO 31
   30 PRINT AT 0,Y;U$;AT 21,Y;U$
   40 NEXT Y
   50 FOR X=0 TO 21
   60 PRINT AT X,0;U$;AT X,31;U$
   70 NEXT X
   80 FOR I=1 TO 200+INT (RND*99)
   82 LET U$="▇"
   85 LET Q=INT (RND*5)
   87 IF Q=1 THEN LET U$=CHR$ 8
   90 LET X=INT (RND*20)+1
  100 LET Y=INT (RND*28)+2
  110 PRINT AT X,Y;U$
  120 NEXT I
  130 PRINT AT 20,31;"S"
  140 PRINT AT 1,0;"›"
  200 LET AF=PEEK 16396+256*PEEK
16397
  210 FOR I=1 TO 726
  220 LET A$(I)=CHR$ PEEK (AF+I)
  230 NEXT I
  235 PRINT AT 1,1;"O"
  240 SLOW
  300 PAUSE 999
  310 CLS
  320 RAND USR 16516
  330 PRINT AT 20,23;"OUF SAUVE"
  400 FOR I=1 TO 726
  405 IF PEEK (AF+I)=27 THEN NEXT
I
  410 POKE AF+I,CODE A$(I)
  430 NEXT I
  440 IF INKEY$="" THEN GOTO 440
  450 RAND
  460 CLS
  470 RUN
  900 FAST
  910 REM RIEN ENTRE LES ""
  920 LET A$=""
```

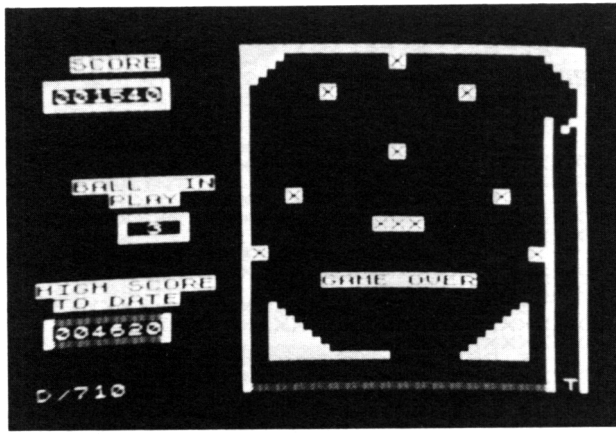
La liste complète des codes de la ligne 1000 à la ligne 1080 est donnée **entière** sur la page suivante.

Ensuite, il faut encore taper les lignes 2200 à 3000 qui assurent le lancement du programme.

```
2000 POKE 16511,19
2010 POKE 16512,1
2200 LET A=1
2210 FOR I=1 TO 236
2220 POKE 16513+I,VAL A$(A TO A+
2)
2230 LET A=A+4
2240 NEXT I
2250 SLOW
2270 POKE 16514,118
2280 POKE 16515,118
2290 REM  SAUVEZ  LE  PROGRAMME
     AVANT DE FAIRE RUN 900
2500 REM ......................
3000 RUN
```

```
1000 LET A$=A$+"000 000 042 012
064 017 035 000 025 034 106 065
042 016 064 017 040 000 025 034
108 065 042 106 065 054 052 042
"
1010 LET A$=A$+"108 065 035 126
254 056 200 017 208 007 027 122
179 032 251 042 106 065 054 027
014 254 006 239 237 120 230 004
"
1020 LET A$=A$+"032 034 042 108
065 035 126 254 008 202 097 065
254 000 032 012 034 108 065 042
106 065 035 034 106 065 024 194
"
1030 LET A$=A$+"042 106 065 035
054 151 024 186 006 247 237 120
230 016 032 029 042 108 065 043
126 254 000 032 012 034 108 065
"
1040 LET A$=A$+"042 106 065 043
034 106 065 024 157 042 106 065
043 054 151 024 149 006 239 237
120 230 008 032 034 042 108 065
"
1050 LET A$=A$+"017 223 255 025
126 254 000 032 013 034 108 065
042 106 065 025 034 106 065 195
152 064 042 106 065 025 054 151
"
1060 LET A$=A$+"195 152 064 006
239 237 120 230 016 032 034 042
108 065 017 033 000 025 126 254
000 032 013 034 108 065 042 106
"
1070 LET A$=A$+"065 025 034 106
065 195 152 064 042 106 065 025
054 151 195 152 064 006 127 237
120 230 001 200 195 152 064 042
"
1080 LET A$=A$+"106 065 035 054
008 195 132 064 000 000 000 000
"
```

FLIPPER




```
                    FLIPPER

DESIREZ VOUS CONNAITRE LE FONC-
TIONNEMENT ?


APRES GAME OVER APPUYEZ SUR "P"
EN MAINTENANT LA TOUCHE

POUR TIRER LA BALLE APPUYEZ SUR
"T"

FLIPPER GAUCHE -> "Z"
FLIPPER DROIT  -> "V"

APPUYEZ SUR UNE TOUCHE...
```

Il est inutile de présenter ici le flipper que connaissent tous les jeunes. Notre programme est très réaliste et surtout très rapide. La course folle des compteurs est exemplaire ; le meilleur score reste affiché.

Il vous faudra beaucoup de maîtrise et de réflexes pour bloquer la bille et la renvoyer.

Un dernier conseil : au cours d'une partie, ne secouez pas trop votre ZX, il pourrait bien faire TILT en perdant la mémoire !

Programme complet Basic

```
   1 REM XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX
   2 REM XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX
   3 REM XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX
   4 REM XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX
   5 REM XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX
 360 PRINT TAB 10;"FLIPPER",,,"D
ESIREZ VOUS CONNAITRE LE FONC- T
IONNEMENT ?"
 370 IF INKEY$="" THEN GOTO 370
 380 IF INKEY$="O" THEN GOSUB 10
00
 390 CLS
 400 PRINT TAB 11;"
"
 410 PRINT TAB 11;"
"
 420 PRINT TAB 11;"
"
 430 PRINT TAB 11;"
"
 440 PRINT TAB 11;"
"
 450 FOR I=1 TO 17
 460 PRINT TAB 11;"
"
 470 NEXT I
 480 LET A=PEEK 16396+256*PEEK 1
6397
```

```
 490 POKE A+771,133
 500 POKE A+738,133
 510 POKE A+756,5
 520 POKE A+758,5
 530 POKE A+789,5
 540 POKE A+791,5
 550 FOR I=772 TO 788
 560 POKE A+I,9
 570 NEXT I
 580 PRINT AT 7,20;"▚";AT 10,14;
"▚";AT 10,26;"▚";AT 12,19;"▚▚▚";
AT 14,12;"▚";AT 14,28;"▚";AT 18,
13;"▙";AT 18,27;"▟";AT 19,13;"▜▛
";AT 19,26;"▜▛";AT 20,13;"▜▛▜▛";A
T 20,25;"▜▛▜▛";AT 21,13;"▜▛▜▛";AT
 21,24;"▜▛▜▛"
 590 PRINT AT 1,2;"SCORE"
 600 PRINT AT 2,0;"          "
 610 PRINT AT 3,0;" 000000 "
 620 PRINT AT 4,0;"          "
 630 PRINT AT 9,2;"BALL  IN";AT
10,4;"PLAY"
 640 PRINT AT 11,4;"      "
 650 PRINT AT 12,4;" 0 "
 660 PRINT AT 13,4;"      "
 670 PRINT AT 16,0;"HIGH SCORE",
," TO BATE"
 680 PRINT "          "
 690 PRINT " 000000 "
 700 PRINT "          "
 710 PRINT AT 16,16;"GAME OVER"
 720 FOR I=1 TO 20
 730 NEXT I
 740 PRINT AT 16,16;"GAME OVER"
 750 FOR I=1 TO 20
 760 NEXT I
 770 IF INKEY$<>"P" THEN GOTO 71
0
 780 PRINT AT 16,16;"          "
 790 PRINT AT 3,1;"000000"
 800 LET B=0
 810 LET B=B+1
```

```
 820 IF B=4 THEN GOTO 900
 830 PRINT AT 12,6;B
 840 POKE A+790,52
 850 POKE A+196,0
 860 IF INKEY$<>"T" THEN GOTO 86
0
 870 POKE A+790,57
 880 RAND USR 16516
 890 GOTO 810
 900 LET A$=""
 910 FOR I=A+101 TO A+106
 920 LET A$=A$+CHR$ PEEK I
 930 NEXT I
 940 LET B$=""
 950 FOR I=A+629 TO A+634
 960 LET B$=B$+CHR$ PEEK I
 970 NEXT I
 980 IF A$>B$ THEN PRINT AT 19,1
;A$
 990 GOTO 710
1000 CLS
1010 PRINT "APRES GAME OVER APPU
YEZ SUR ""P"" EN MAINTENANT LA T
OUCHE",,,"POUR TIRER LA BALLE AP
PUYEZ SUR ""T"" "
1020 PRINT ,,"FLIPPER GAUCHE -> 
""Z"" ","FLIPPER DROIT  -> ""U""
 "
1030 PRINT ,,,,"APPUYEZ SUR UNE
TOUCHE..."
1040 IF INKEY$="" THEN GOTO 1040
1050 RETURN
1060 FAST
1064 REM RIEN ENTRE LES ""
1065 LET A$=""
```

```
1070 LET A$=A$+"000 000 024 006
000 000 000 000 000 000 042 012
064 001 245 002 009 034 134 064
033 223 255 034 136 064 042 012
"

1080 LET A$=A$+"064 001 199 002
009 034 191 065 001 004 000 009
034 193 065 033 000 000 034 138
064 205 184 064 024 251 042 134
"

1090 LET A$=A$+"064 054 000 237
075 136 064 009 062 189 190 032
011 006 009 205 171 065 005 120
254 000 032 247 042 134 064 237
"

1100 LET A$=A$+"075 136 064 009
062 000 190 040 116 062 009 190
202 195 065 001 254 247 237 120
230 016 202 195 065 205 171 065
"

1110 LET A$=A$+"237 095 130 178
020 203 191 203 183 203 175 203
167 254 006 048 239 254 000 032
003 001 224 255 254 001 032 003
"

1120 LET A$=A$+"001 222 255 254
002 032 003 001 032 000 254 003
032 003 001 034 000 254 004 032
003 001 255 255 254 005 032 003
"

1130 LET A$=A$+"001 001 000 254
006 032 003 001 033 000 254 007
032 003 001 223 255 237 067 136
064 042 012 064 213 017 196 000
"

1140 LET A$=A$+"025 054 006 209
033 032 003 034 138 064 195 210
064 054 052 034 134 064 001 254
247 237 120 230 016 040 098 042
"

1150 LET A$=A$+"138 064 017 120
000 025 034 138 064 043 125 180
032 251 042 191 065 054 000 035
054 000 035 054 000 035 035 054
"
```

```
1160 LET A$=A$+"000 035 054 000
035 054 000 001 254 254 237 120
230 002 032 012 042 191 065 054
131 035 054 131 035 054 131 201
"
1170 LET A$=A$+"237 120 230 016
192 042 193 065 054 131 035 054
131 035 054 131 201 042 012 064
017 105 000 025 126 254 037 040
"
1180 LET A$=A$+"003 060 119 201
054 028 043 024 243 000 000 000
000 225 201 "

2000 POKE 16511,110
2010 POKE 16512,1
2200 LET A=1
2210 FOR I=1 TO 323
2220 POKE 16513+I,VAL A$(A TO A+
2)
2230 LET A=A+4
2240 NEXT I
2250 SLOW
2270 POKE 16514,118
2280 POKE 16515,118
2290 REM SAUVEZ LE PROGRAMME
AVANT DE FAIRE RUN 1060
2500 REM ......................
3000 RUN
```

CASSE briques

Qui ne connaît ce jeu fort simple où il s'agit de « descendre » plusieurs rangées de briques alignées à l'aide du plus petit nombre de balles possible et, tout de même, d'une raquette que le joueur déplace sur l'écran ?

La balle rebondit de tous les côtés sur la raquette, bien sûr, mais également sur les briques et les bords du jeu. Il faut un bon coup d'œil pour anticiper sur la trajectoire de la balle et des réflexes rapides pour placer la raquette.

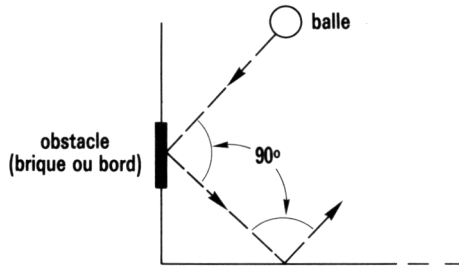
Quel spectacle lorsqu'une brèche est enfin ouverte dans le mur de briques ! Surtout si l'on parvient à y engouffrer la balle qui démolit à toute allure les rangées arrière par de multiples et rapides rebonds.

Seul le langage machine peut nous offrir de telles possibilités.

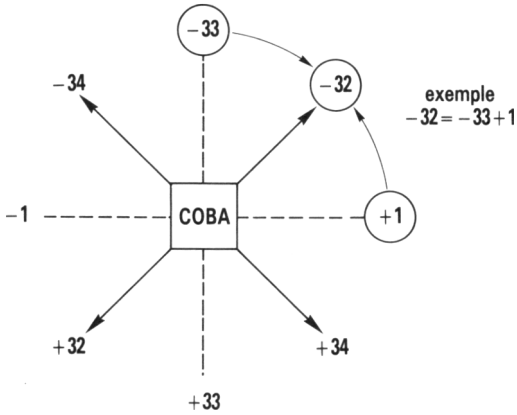
Avant de vous présenter plus avant le détail du programme, il nous semble indispensable de donner quelques explications sur le déplacement de la balle et de la raquette.

Balle

La balle se déplace dans 4 directions obliques seulement, les rebonds se faisant à 90°.



Pour déplacer la balle, il suffit d'ajouter à sa position sur l'écran (COBA = coordonnées balle dans le programme assembleur) la valeur de la variable DIREC, cette variable prenant les valeurs : +32, −32, +34, −34 selon la direction choisie.



La variable DIREC est formée par la combinaison, ou plutôt la somme algébrique des directions horizontales (DIR 2) et verticales (DIR 1).
DIR 1 vaudra −33 ou +33
DIR 2 vaudra − 1 ou + 1

Le petit dessin suivant résume ce principe :

DIREC = −33 −1 = −34	ligne supérieure DIR1 = −33	DIREC = −33 + 1 = −32
position précédente DIR 2 = −1	position initiale COBA	position suivante DIR 2 = +1
DIREC = −33 −1 = +32	ligne inférieure DIR 1 = +33	DIREC = −33 + 1 = +34

Rebonds

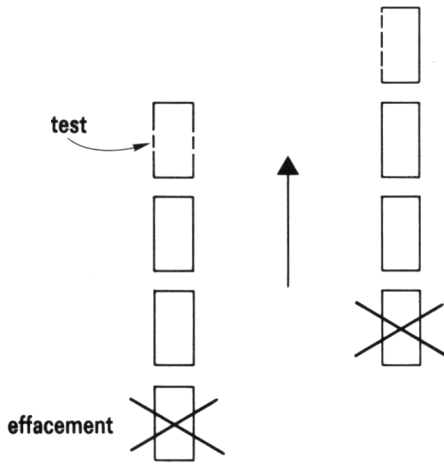
Le principe là aussi est fort simple. Chaque fois qu'il y aura un obstacle au-dessus ou au-dessous de la balle, il suffira de changer le signe de DIR 1 (vertical). Pour les obstacles à droite ou à gauche, il convient bien sûr de changer le signe de DIR 2 (horizontal).

Dans le programme assembleur, ce travail est respectivement assuré par les sous-programmes CAS 1 et CAS 2.

Raquette

Elle sera représentée par 3 caractères contigus d'une même colonne. Pour opérer un déplacement vers le haut, il faut tester si la case visée est bien vide. Si c'est le cas, on affiche le caractère dans cette case et on efface le caractère du bas de la raquette. Le même système sera utilisé vers le bas.

Exemple :



Comme pour le chronomètre digital, nous vous suggérons de photocopier, puis d'assembler les pages regroupant le programme en langage d'assemblage afin de bien distinguer les différentes parties et sous-programmes.

Pour jouer, la touche N déplace la raquette vers le haut si cela est possible ; de même pour la touche V qui opère un déplacement de raquette vers le bas (1).

Algorithme du programme

1) Initialisations :

— Coordonnées de la raquette.
— Coordonnée de la balle.
— Direction 1 (verticale).
— Direction 2 (horizontale).
— Durée de la temporisation.

2) Test des cases autour de la balle :

— Si la case au-dessus ou en dessous est un bord, alors changer de direction 1.
— Si c'est une brique, alors changer de direction 1, effacer la brique et incrémenter le score.
— Si la case à gauche est une brique, alors changer de direction 2, effacer la brique et incrémenter le score.
— Si c'est un caractère 118, alors changer de direction 2.
— Si la case à droite est une raquette, alors changer de direction 2.
— Si c'est un caractère 118, alors revenir au Basic.

3) Test de la case suivante :

— Si c'est une brique, alors changer de direction 1 et 2, effacer la brique et incrémenter le score.
— Si c'est une raquette, alors changer de direction 1 et 2.

4) Effacement de l'ancienne balle, affichage de la nouvelle, temporisation.

5) Test des touches N et V :

— Si la touche N est enfoncée, alors déplacer la raquette vers le haut si possible.
— Si la touche V est enfoncée, alors déplacer la raquette vers le bas si possible.

6) Boucle du test des cases autour de la balle.

---

(1) Quelques nouvelles instructions ont été utilisées ici, à savoir NEG, PUSH, POP. Nous vous invitons à les retrouver dans l'aide-mémoire situé à la fin de l'ouvrage.

Programme en langage d'assemblage

PARTIE 1

```
        ;INITIALISATION
        LD HL,(+16396)
        LD DE,+97
        ADD HL,DE
        LD (CORA),HL
        LD HL,(+16396)
        LD DE,+95
        ADD HL,DE
        LD (COBA),HL
        LD HL,+33
        LD (DIR1),HL
        LD HL,-1
        LD (DIR2),HL
        LD BC,+0000
        LD (TEMPO),BC

DEBUT   LD HL,(COBA)
        LD DE,-33
        ADD HL,DE
        LD A,+138
        CP(HL)
        CALL Z CAS1
        LD A,5
        CP(HL)
        JR NZ LAB1
        CALL CAS1
        CALL EFFAC

LAB1    LD DE,+34
        ADD HL,DE
        LD A,+118
        CP(HL)
        JR NZ SUIT
        LD HL,(COBA)
        LD (HL),+128
        RET
SUIT    LD A,+133
        CP(HL)
        CALL Z CAS2
        LD A,5
        CP(HL)
        JR NZ LAB2
        CALL CAS2
        CALL EFFAC
```

PARTIE 2

```
LAB2    LD DE,+32
        ADD HL,DE
        LD A,+137
        CP(HL)
        CALL Z CAS1
        LD A,5
        CP(HL)
        JR NZ LAB3
        CALL CAS1
        CALL EFFAC

LAB3    LD DE,-34
        ADD HL,DE
        LD A,+118
        CP(HL)
        CALL Z CAS2
        LD A,5
        CP(HL)
        JR NZ LAB4
        CALL CAS2
        CALL EFFAC

LAB4    LD DE,(DIR1)
        LD HL,(DIR2)
        ADD HL,DE
        LD D,H
        LD E,L
        LD (DIREC),DE
        LD HL,(COBA)
        ADD HL,DE
        LD A,5
        CP(HL)
        JR NZ LAB6
        CALL CAS1
        CALL CAS2
        CALL EFFAC
        JP DEBUT
LAB6    LD A,+128
        CP(HL)
        JR Z LAB5
        CALL CAS1
        CALL CAS2
        JP DEBUT

        ;TEMPORISATION
LAB5    LD BC,(TEMPO)
ICI     DEC BC
        LD A,B
        OR C
        JR NZ ICI
```

PARTIE 3

```
        ;EFFACEMENT BALLE
        LD HL,(COBA)
        LD(HL),+128
        ;CALCUL ET AFFICH.BALLE
        LD DE,(DIREC)
        ADD HL,DE
        LD(COBA),HL
        LD(HL),+180

        ;TEST TOUCHE RAQUETTE
        LD C,+254
        LD B,+127;TOUCHE N
        IN A,(C)
        AND 8
        JR NZ LBL1
        LD HL,(CORA)
        LD DE,-33
        ADD HL,DE
        LD A,+138
        CP(HL)
        JR Z LBL1
        LD (CORA),HL
        LD(HL),+133
        LD DE,+99
        ADD HL,DE
        LD(HL),+128

LBL1    LD B,+254;TOUCHE V
        IN A,(C)
        AND +16
        JP NZ DEBUT
        LD HL,(CORA)
        LD DE,+99
        ADD HL,DE
        LD A,+137
        CP(HL)
        JP Z DEBUT
        LD HL,(CORA)
        LD(HL),+128
        LD DE,+33
        ADD HL,DE
        LD (CORA),HL
        LD DE,+66
        ADD HL,DE
        LD (HL),+133

        JP DEBUT

        ;SOUS PROGRAMMES
```

```
        ;CHANGEMENT DE DIRECTION1
CAS1    LD DE,(DIR1)
        LD A,E
        NEG
        LD E,A
        LD A,D
        CPL
        LD D,A
        LD (DIR1),DE
        RET
        ;CHANGEMENT DE DIRECTION2
CAS2    LD DE,(DIR2)
        LD A,E
        NEG
        LD E,A
        LD A,D
        CPL
        LD D,A
        LD(DIR2),DE
        RET
        ;EFFACEMENT BRIQUE
EFFAC   LD(HL),+128
        PUSH HL
        LD DE,-10
        LD HL,(TEMPO)
        ADD HL,DE
        LD(TEMPO),HL
        ;INCREMENTATION SCORE
        LD HL,(+16396)
        LD DE,+770
        ADD HL,DE
XX1     LD A,(HL)
        CP+37
        JR Z XX2
        INC A
        LD(HL),A
        POP HL
        RET
XX2     LD(HL),+28
        DEC HL
        JR XX1

        ;VARIABLES
COBA    0000
CORA    0000
DIR1    0000
DIR2    0000
TEMPO   0000
DIREC   0000
```

Programme complet Basic

```
    1 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    2 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    3 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    4 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    5 REM XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  104 CLS
  105 PRINT TAB 10;"CASSE-BRIQUE"
,,,
   30 PRINT "COMBIEN DE BALLES VO
ULEZ VOUS ?",,,
  107 INPUT N
  108 PRINT "CHOISISSEZ LA VITESS
E DU JEU (1 A 9) "
  109 INPUT V
  110 LET V=3000-V*222
  111 POKE 16549,V-256*INT (V/256
)
  112 POKE 16550,INT (V/256)
  113 CLS
  119 PRINT "
                                  ",
  120 FOR I=1 TO 21
  130 PRINT "
                                  "
  140 NEXT I
  150 POKE 16442,4
  160 PRINT "
                                  ",
  170 PRINT "  SCORE 0000
                                  "
```

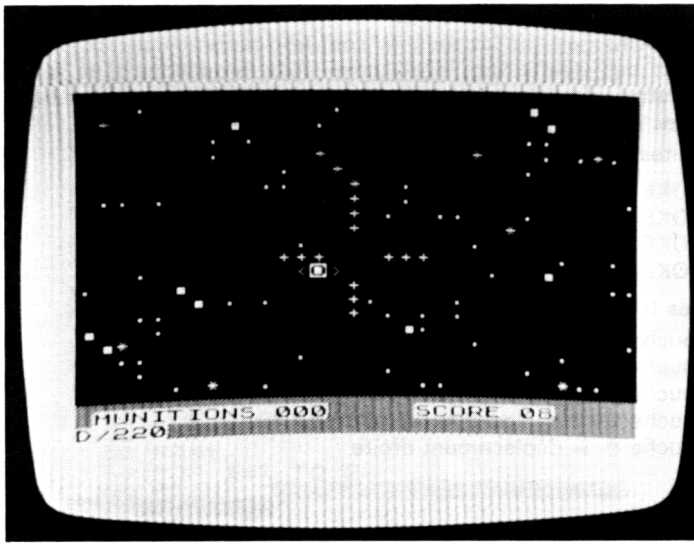
```
 500 LET AF=PEEK 16396+256*PEEK
16397
 505 FOR J=1 TO N
 510 LET I=97
 520 POKE AF+I,133
 521 POKE 33+AF+I,133
 522 POKE 66+AF+I,133
 523 IF INKEY$="" THEN GOTO 523
 525 RAND
 530 POKE 16530,INT (RND*11)+85
 600 RAND USR 16516
 610 FOR I=64 TO 724 STEP 33
 620 POKE AF+I,128
 630 NEXT I
 640 NEXT J
 650 PRINT AT 10,13;"GAME OVER"
 660 IF CODE INKEY$=118 THEN RUN

 670 GOTO 660

 800 REM
 900 REM RIEN ENTRE LES ""
 910 LET A$=""
 920 FAST
 950 REM

1000 LET A$=A$+"118 118 042 012
064 017 097 000 025 034 230 065
042 012 064 017 095 000 025 034
228 065 033 033 000 034 232 065
"

1010 LET A$=A$+"033 255 255 034
234 065 001 000 000 237 067 236
065 042 228 065 017 223 255 025
062 138 190 204 162 065 062 005
"

1020 LET A$=A$+"190 032 006 205
162 065 205 194 065 017 034 000
025 062 118 190 032 006 042 228
065 054 128 201 062 133 190 204
"

1030 LET A$=A$+"178 065 062 005
190 032 006 205 178 065 205 194
065 017 032 000 025 062 137 190
204 162 065 062 005 190 032 006
"

1040 LET A$=A$+"205 162 065 205
194 065 017 222 255 025 062 118
190 204 178 065 062 005 190 032
006 205 178 065 205 194 065 237
"
```

```
1050 LET A$=A$+"091 232 065 042
234 065 025 084 093 237 083 238
065 042 228 065 025 062 005 190
032 012 205 162 065 205 178 065
"
1060 LET A$=A$+"205 194 065 195
171 064 062 128 190 040 009 205
162 065 205 178 065 195 171 064
237 075 236 065 011 120 177 032
"
1070 LET A$=A$+"251 042 228 065
054 128 237 091 238 065 025 034
228 065 054 180 014 254 006 127
237 120 250 008 032 023 042 230
"
1080 LET A$=A$+"065 017 223 255
025 062 138 190 040 011 034 230
065 054 133 017 099 000 025 054
128 006 254 237 120 230 016 194
"
1090 LET A$=A$+"171 064 042 230
065 017 099 000 025 062 137 190
202 171 064 042 230 065 054 128
017 033 000 025 034 230 065 017
"
1100 LET A$=A$+"066 000 025 054
133 195 171 064 237 091 232 065
123 237 068 095 122 047 087 237
083 232 065 201 237 091 234 065
"
1110 LET A$=A$+"123 237 068 095
122 047 087 237 083 234 065 201
054 128 229 017 246 255 042 236
065 025 034 236 065 042 012 064
"
1120 LET A$=A$+"017 002 003 025
126 254 037 040 004 060 119 225
201 054 028 043 024 242 000 000
000 000 000 000 000 000 000 000
"
1130 LET A$=A$+"000 000 000 000
000 "
```

```
1999 REM
2000 POKE 16511,205
2010 POKE 16512,1
2020 LET A=1
2210 FOR I=1 TO 369
2220 POKE 16513+I,VAL A$(A TO A+
2)
2230 LET A=A+4
2240 NEXT I
2250 SLOW
2270 POKE 16514,118
2280 POKE 16515,118
2290 REM SAUVEZ LE PROGRAMME AVA
NT DE FAIRE RUN 900
2500 REM ....................
3000 RUN
```

DÉFENSE

Spatiale



Aux confins de la galaxie, votre mission consiste à repousser les assaillants qui rôdent sans cesse autour de votre base.

Le radar de poursuite se positionnera automatiquement sur le vaisseau ennemi qui cherche à vous contourner et le maintiendra dans votre champ de vision. Il est d'ailleurs possible de le suivre dans **toutes** les directions pour parvenir à l'ajuster dans le viseur de votre quadruple laser meurtrier.

Les déplacements conjugués des étoiles, du vaisseau et de vous-même rendent ce combat particulièrement vertigineux.

L'éclair aveuglant du laser frappe de plein fouet l'ennemi qui explose et dont les débris se volatilisent dans les ténèbres spatiales.

Mais déjà un autre danger s'approche ne vous laissant aucun répit.

Les touches de déplacement du clavier vous aideront véritablement à évoluer dans les 3 dimensions ; la touche 4 sera réservée au déclenchement du tir qui décrémente rapidement le compteur des munitions.

Si vous jugez que les touches sont trop rapprochées ou... si vous avez des doigts trop gros, nous vous proposons d'en utiliser d'autres plus espacées en procédant ainsi en mode d'exécution directe.

Faites

```
POKE 16546, 2
POKE 16553, 8
POKE 16568, 4
POKE 16592, 1
```

Les touches auront à présent les attributions suivantes :

touche 2 = tir
touche 4 = déplacement gauche
touche 6 = déplacement bas
touche 8 = déplacement haut
touche Ø = déplacement droite

```
   1 REM XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   2 REM XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   3 REM XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   4 REM XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   5 REM XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   6 REM XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   7 REM XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   8 REM XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   9 REM XXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  10 DIM A$(693)
  15 RAND
  20 LET A=PEEK 16396+PEEK 16397
*256
  30 FAST
  40 FOR I=1 TO 20
  50 PRINT "                    
                    "
  60 NEXT I
  70 FOR I=1 TO 60
  80 PRINT AT RND*19,RND*31;"▝"
  90 NEXT I
  92 FOR I=1 TO 10
  94 PRINT AT RND*19,RND*31;"▛"
  95 PRINT AT RND*19,RND*31;"▜"
  96 NEXT I
 100 FOR I=1 TO 693
 110 LET A$(I)=CHR$ PEEK (A+I)
 120 NEXT I
 130 CLS
 140 PRINT AT 0,0;"                    
                    "
 150 PRINT AT 21,0;"                    
                    "
 160 SLOW
 170 POKE 16442,4
 180 PRINT "▝MUNITIONS 250       S
CORE 00          "
```

```
200 RAND USR 16516
202 PAUSE 1000
210 IF INKEY$="" THEN GOTO 210
220 GOTO 130
900 REM RIEN ENTRE LES ""
910 LET A$=""
920 FAST
```

Vous trouverez les codes à introduire dans les lignes 1000 à 1270 des pages suivantes.

Puis, il restera à taper le petit programme de lancement ci-après :

```
2000 POKE 16511,65
2010 POKE 16512,3
2200 LET A=1
2210 FOR I=1 TO 780
2220 POKE 16513+I,VAL A$(A TO A+
2)
2230 LET A=A+4
2240 NEXT I
2250 SLOW
2270 REM SAUVEZ LE PROGRAMME
         AVANT DE FAIRE RUN 900
2500 REM ......................
3000 RUN
```

```
1000 LET A$=A$+"118 118 062 250
050 131 067 205 248 064 042 012
064 001 104 000 009 034 139 067
205 094 065 205 185 065 001 254
"
1010 LET A$=A$+"247 237 120 230
008 204 182 066 237 120 230 016
032 006 205 085 066 205 185 065
001 254 239 237 120 230 008 032
"
1020 LET A$=A$+"006 205 247 065
205 185 065 237 120 230 016 032
006 205 038 066 205 185 065 237
120 230 004 032 006 205 135 066
"
1030 LET A$=A$+"205 185 065 000
237 091 133 067 027 237 083 133
067 122 179 204 248 064 237 075
135 067 042 139 067 009 034 139
"
1040 LET A$=A$+"067 205 185 065
024 091 058 052 064 071 237 095
017 005 000 237 083 133 067 128
060 203 191 203 183 203 175 203
"
1050 LET A$=A$+"167 254 009 048
229 001 000 000 254 001 032 003
001 032 000 254 002 032 003 001
224 255 254 003 032 003 001 033
"
1060 LET A$=A$+"000 254 004 032
003 001 223 255 254 005 032 003
001 034 000 254 006 032 003 001
222 255 254 007 032 003 001 001
"
1070 LET A$=A$+"000 254 008 032
003 001 255 255 237 067 135 067
201 017 208 007 027 122 179 032
251 195 150 064 042 012 064 001
"
```

```
1080 LET A$=A$+"034 000 009 084
093 042 016 064 001 006 000 009
001 148 002 237 176 042 139 067
054 052 043 054 147 035 035 054
"

1090 LET A$=A$+"146 042 012 064
001 247 000 009 054 149 001 033
000 009 054 149 009 054 149 001
062 000 009 054 149 035 054 149
"

1100 LET A$=A$+"035 054 149 001
004 000 009 054 149 035 054 149
035 054 149 001 062 000 009 054
149 001 033 000 009 054 149 009
"

1110 LET A$=A$+"054 149 201 042
139 067 043 043 126 254 118 032
006 205 085 066 205 094 065 001
004 000 009 126 254 118 032 006
"

1120 LET A$=A$+"205 135 066 205
094 065 001 221 255 009 126 254
131 032 006 205 247 065 205 094
065 001 066 000 009 126 254 137
"

1130 LET A$=A$+"032 006 205 038
066 205 094 065 201 042 016 064
001 187 002 009 084 093 001 223
255 009 001 149 002 237 184 042
"

1140 LET A$=A$+"016 064 001 006
000 009 084 093 001 148 002 009
001 033 000 237 176 042 139 067
001 033 000 009 034 139 067 201
"

1150 LET A$=A$+"042 016 064 001
154 002 009 084 093 001 108 253
009 001 032 000 237 176 042 016
064 001 006 000 009 084 093 001

1160 LET A$=A$+"033 000 009 001
148 002 237 176 042 139 067 001
223 255 009 034 139 067 201 033
020 000 034 137 067 042 016 064
"

1170 LET A$=A$+"001 037 000 009
084 093 126 043 001 031 000 237
184 018 237 091 137 067 027 237
083 137 067 122 179 040 006 001
"

1180 LET A$=A$+"065 000 009 024
227 042 139 067 035 034 139 067
201 033 020 000 034 137 067 042
016 064 001 006 000 009 084 093
"

1190 LET A$=A$+"126 035 001 031
000 237 176 018 237 091 137 067
027 237 083 137 067 122 179 040
003 035 024 230 042 139 067 043
"
```

```
1200 LET A$=A$+"034 139 067 201
042 012 064 017 039 000 025 006
005 017 034 000 054 000 025 016
251 006 005 054 006 025 016 251
"
1210 LET A$=A$+"042 012 064 017
059 000 025 017 032 000 006 005
054 000 025 016 251 006 005 054
134 025 016 251 042 012 064 017
"
1220 LET A$=A$+"155 002 025 017
224 255 006 004 054 000 025 016
251 006 005 054 134 025 016 251
042 012 064 017 173 002 025 017
"
1230 LET A$=A$+"222 255 006 004
054 000 025 016 251 006 005 054
006 025 016 251 054 189 237 091
139 067 124 146 254 000 032 006
"
1240 LET A$=A$+"125 147 254 000
040 037 042 012 064 001 228 002
009 126 254 028 040 004 061 119
024 006 062 037 119 043 024 241
"
1250 LET A$=A$+"058 131 067 061
050 131 067 254 000 192 205 094
065 225 201 006 050 197 205 094
065 193 062 255 061 254 000 032
"
1260 LET A$=A$+"251 042 012 064
017 122 001 025 054 019 035 054
180 035 054 018 016 227 225 042
012 064 017 241 002 025 126 254
"
1270 LET A$=A$+"037 040 005 060
119 195 137 064 054 028 043 024
241 000 000 000 000 000 000 000
000 000 000 000 "
```

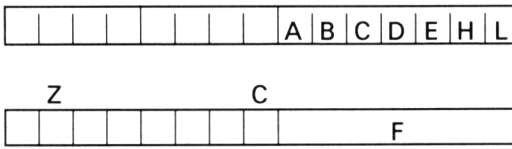
AIDE

mémoire

Dans ce chapitre, nous vous proposons un rappel des instructions du Z 80 vues dans les exemples et jeux précédents. Chacune d'entre elles sera suivie à nouveau d'une brève explication sur le travail qu'elle effectue, d'une liste des instructions équivalentes (qui font le même travail dans des registres différents) et, bien sûr, de leurs codes. Rappelons que la liste des mnémoniques présentées ne sera pas exhaustive, mais largement suffisante pour pouvoir créer de très belles réalisations.

• Rappel des registres

A (accumulateur), B, C, D, E, F (Flag ou indicateur), H, et L sont tous des registres sur 8 bits ou 1 octet.



On peut associer les registres B et C, D et E, H et L pour créer des paires de registres sur 16 bits ou 2 octets.

• Liste des mnémoniques

Nous allons prendre comme convention :
d : donnée sur 8 bits c'est-à-dire comprise entre Ø et 255 (inclus)
dd : donnée sur 16 bits, comprise entre Ø et 65535 (inclus)
nn : adresse sur 16 bits

dd et nn sont à décomposer en Octet Plus Significatif et Octet Moins Significatif, d'après la formule suivante :

OMS = dd − 256 X INT (dd/256)

OPS = INT (dd/256)

LD paire de registres (nn)
charge automatiquement dans la paire de registres, le contenu de l'adresse nn plus 256 fois le contenu de l'adresse nn + 1.
(c.à.d. PEEK nn + 256 X PEEK (nn + 1))

LD HL,(nn)	42, OMS, OPS de nn
LD BC,(nn)	237, 75, OMS, OPS
LD DE,(nn)	237, 91, OMS, OPS

LD A,(nn) 58, OMS, OPS de nn
charge dans l'accumulateur, uniquement le contenu de l'adresse nn.

LD A, (paire de registres)
charge dans l'accumulateur, le contenu de l'adresse représentée par la paire de registres.

LD A,(HL)	126
LD A,(BC)	1Ø
LD A,(DE)	26

LD (HL),d ou registre
charge le contenu de l'adresse représentée par HL, la valeur d ou celle du registre (c.à.d. POKE HL, d).

LD (HL),d	54, d	LD (HL),D	114
LD (HL),A	119	LD (HL),E	115
LD (HL),B	112	LD (HL),H	116
LD (HL),C	113	LD (HL),L	117

LD (nn), paire de registres
charge le contenu de l'adresse nn et le contenu de l'adresse nn + 1, la valeur de la paire de registres.

LD (nn),HL	34, OMS, OPS de nn
LD (nn),BC	237, 67, OMS, OPS
LD (nn),DE	237, 83, OMS, OPS

LD (nn), A 50, OMS, OPS de nn
charge le contenu de l'adresse nn, la valeur de A.

LD registre 1, registre 2 ou d
charge dans le registre 1, la valeur d ou celle du registre 2.

LD A,d	62,d	LD B,d	6,d	LD C,d	14,d	LD D,d	22,d
LD A,B	12Ø	LD B,A	71	LD C,A	79	LD D,A	87
LD A,C	121	LD B,C	65	LD C,B	72	LD D,B	80
LD A,D	122	LD B,D	66	LD C,D	74	LD D,C	81
LD A,E	123	LD B,E	67	LD C,E	75	LD D,E	83
LD A,H	124	LD B,H	68	LD C,H	76	LD D,H	84
LD A,L	125	LD B,L	69	LD C,L	77	LD D,L	85

LD E,d	3Ø,d	LD H,d	38,d	LD L,d	46,d
LD E,A	95	LD H,A	1Ø3	LD L,A	111
LD E,B	88	LD H,B	96	LD L,B	1Ø4
LD E,C	89	LD H,C	97	LD L,C	1Ø5
LD E,D	9Ø	LD H,D	98	LD L,D	1Ø6
LD E,H	92	LD H,E	99	LD L,E	1Ø7
LD E,L	93	LD H,L	1Ø1	LD L,H	1Ø8

LD A,R 237, 95 peut être intéressant dans la mesure où R est utilisé par la machine. LD A,R charge A avec une valeur plus ou moins aléatoire.

LD paire de registre, dd
charge dans la paire de registres la valeur dd.

LD HL,dd 33, OMS, OPS de dd
LD BC,dd 1, OMS, OPS
LD DE,dd 17, OMS, OPS

Notez que l'instruction LD paire de registres 1, paire de registres 2 n'existe pas. Pour effectuer, par exemple, LD HL,DE, il faut faire successivement
LD H,D
LD L,E

INC registre ou paire de registres
incrémente d'une unité la valeur du registre ou celle de la paire de registres.

INC A	6Ø	INC E	28	INC BC	3
INC B	4	INC H	36	INC DE	19
INC C	12	INC L	44	INC HL	35
INC D	2Ø				

DEC registre ou paire de registres
décrémente d'une unité la valeur du registre ou celle de la paire de registres.

DEC A	61	DEC E	29	DEC BC	11
DEC B	5	DEC H	37	DEC BC	27
DEC C	13	DEC L	45	DEC BC	43
DEC D	21				

ADD A,d ou registre
ajoute à A la valeur d ou celle du registre.

ADD A,d	198, d	ADD A,D	13Ø
ADD A,A	135	ADD A,E	131
ADD A,B	128	ADD A,H	132
ADD A,C	129	ADD A,L	133

ADD HL, paire de registres
ajoute à HL la valeur de la paire de registres.

ADD HL,BC	9
ADD HL,DE	25
ADD HL,HL	41

Notez que l'instruction ADD HL,dd n'existe pas. Il suffit de faire successivement :
LD DE,dd
ADD HL,DE

DJNZ d 16, d
décrémente d'une unité le registre B, puis effectue un saut relatif si B est **non** nul. Si B est nul, l'instruction suivante est exécutée. Le déplacement peut être au maximum de +127 vers l'avant et au maximum de −128 vers l'arrière : dans ce cas d=256 — valeur absolue du déplacement.

Remarque : ici NZ n'a rien à voir avec l'indicateur Z du registre F.

CP d ou registre
compare l'accumulateur A avec la valeur d ou celle du registre.
L'indicateur Z du registre F se positionnera à 1
si A=d ou A=registre
L'indicateur Z du registre se positionnera à Ø
si A< >d ou A< >registre
CP agit également sur l'indicateur de report C du registre F :
L'indicateur C se positionnera à 1 si A<d ou A<registre
L'indicateur C se positionnera à Ø si A≥d ou A≥registre.

CP d	254, d	CP D	186
CP A	191	CP E	187
CP B	184	CP H	188
CP C	185	CP L	189

CP (HL) 19Ø, compare A avec la valeur contenue dans l'adresse représentée par HL.

JR d 24,d
effectue un saut relatif inconditionnel. Comme pour DJNZ le déplacement doit être compris entre −128 et +127. Le déplacement se code comme pour DJNZ.

JR condition, d
effectue un saut relatif conditionnel.
JR Z,d 4Ø,d n'effectue le saut que si Z=1
JR NZ,d 32,d n'effectue le saut que si Z=Ø
JR C,d 56,d n'effectue le saut que si C=1
JR NZ,d 48,d n'effectue le saut que si C=Ø
sinon l'instruction est ignorée.

Remarque : les sauts relatifs sont faciles à utiliser lorsque le déplacement n'est pas trop grand (compris entre −1Ø et +1Ø environ). Si ce dernier est trop important ou à chaque fois qu'il doit être supérieur à 128, il faudra utiliser l'instruction JP nn.

JP nn 195, OMS, OPS de nn
effectue un saut inconditionnel à l'adresse nn.

JP condition, nn
effectue un saut conditionnel à l'adresse nn.

JP Z,nn	2Ø2, OMS, OPS de nn n'effectue le saut que	si Z=1
JP NZ,nn	194, OMS, OPS de nn n'effectue le saut que	si Z=Ø
JP C,nn	218, OMS, OPS de nn n'effectue le saut que	si C=1
JP NC,nn	21Ø, OMS, OPS de nn n'effectue le saut que	si C=Ø

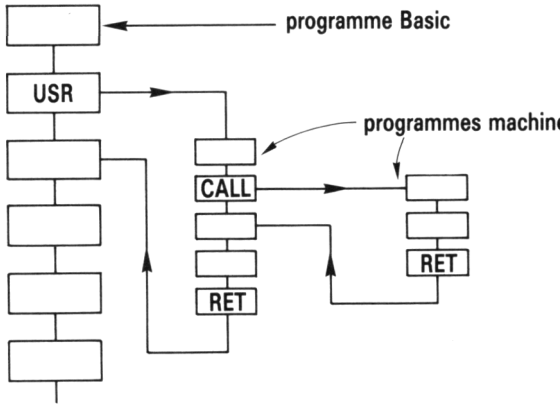
sinon l'instruction est ignorée.

CALL nn 2Ø5, OMS, OPS de nn
appelle un sous-programme à partir de l'adresse nn.

CALL condition, nn

CALL Z,nn	2Ø4, OMS, OPS de nn appelle le sous-programme si Z=1
CALL NZ,nn	196, OMS, OPS de nn appelle le sous-programme si Z=Ø
CALL C,nn	22Ø, OMS, OPS de nn appelle le sous-programme si C=1
CALL NC,nn	212, OMS, OPS de nn appelle le sous-programme si C=Ø

RET 201
effectue un retour au programme principal (celui qui a appelé le sous-programme). Si aucun CALL n'a été effectué, RET provoque le retour au Basic, à la ligne suivant immédiatement l'instruction USR. Si un CALL a été effectué, RET provoque le retour à l'instruction suivant immédiatement CALL.



RET condition

RET Z	2ØØ,	effectue le retour si Z=1
RET NZ	192,	effectue le retour si Z=Ø
RET C	216,	effectue le retour si C=1
RET NC	2Ø8,	effectue le retour si C=Ø

OR d ou registre

effectue l'opération logique OU entre A et d, ou entre A et un registre, le résultat est rangé dans A, et l'indication Z est forcée à 1 si A=Ø, à Ø si A<>Ø.

OR d	246, d	OR D	178
OR A	183	OR E	179
OR B	176	OR H	18Ø
OR C	177	OR L	181

AND d ou registre

effectue l'opération logique ET entre A et d, ou entre A et le registre, le résultat est rangé dans A, Z est forcée à 1 si A=Ø, à Ø si A<>Ø.

AND d	23Ø, d	AND D	162
AND A	167	AND E	163
AND B	16Ø	AND H	164
AND C	161	AND L	165

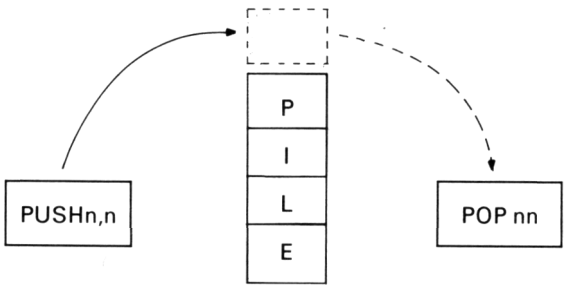
NOP Ø
n'effectue rien du tout. NOP peut servir à réserver de la place en mémoire, ou à supprimer des instructions.

• Remarque sur les nombres négatifs

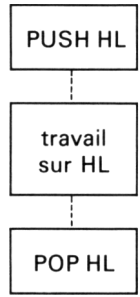
Nous avons vu que sur 8 bits nous pouvions représenter des nombres entre Ø et 255 et que sur 16 bits ils pouvaient être compris entre Ø et 65535.
Voyons comment représenter des nombres négatifs :
— sur 8 bits, les nombres négatifs peuvent aller jusqu'à −128 au maximum. d sera codé de la façon suivante : d=256 − la valeur absolue du nombre. Ceci est lié au fait que sur 8 bits, 255+1 donne Ø. 255 peut donc être considéré comme −1 ; de même 254+2 donne Ø, à 254 correspond donc −2, etc.
— sur 16 bits, les nombres négatifs peuvent aller jusqu'à −32768 au maximum. Cette fois-ci, dd=65536 − la valeur absolue du nombre.
Même explication : 65535+1 donne Ø, et 65535 correspond à −1,
65534+2 donne Ø, et 65534 correspond à −2,
etc.

NEG 237, 68
Cette instruction transforme la valeur de l'accumulateur ou registre A en son opposé.

PUSH paire de registres, POP paire de registres
L'instruction PUSH empile la valeur de la paire de registres sur la « PILE ». La pile est la zone mémoire correspondant à **pile machine** (voir manuel p. 171). L'instruction POP enlève de la pile la **dernière** valeur empilée et la charge dans la paire de registres suivant POP.

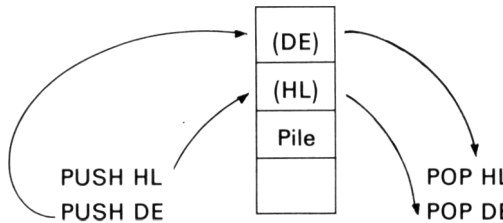


PUSH et POP permettent donc de mémoriser un instant la valeur d'une paire de registres pour pouvoir travailler sur cette paire. Exemple :



Ces instructions permettent aussi d'échanger les valeurs de 2 paires de registres :

```
PUSH  HL
PUSH  DE
POP   HL
POP   DE
```



Attention ! Il faut veiller scrupuleusement à faire correspondre à chaque PUSH un POP pour ne pas créer des erreurs de pile. Cette remarque est primordiale dans un sous-programme en langage machine, car l'instruction CALL empile, elle, l'adresse de retour qui sera lue par RET.

PUSH BC	197	POP BC	1Ø3
PUSH DE	213	POP DE	2Ø9
PUSH HL	229	POP HL	225

La spirale infernale : solution

Voici les codes et adresses correspondantes :

16514	42
16515	12
16516	64
16517	17
16518	34
16519	0
16520	25
16521	54
16522	52
16523	17
16524	205
16525	7
16526	27
16527	122
16528	179
16529	32
16530	251
16531	54
16532	0
16533	14
16534	254
16535	6
16536	247
16537	237
16538	120
16539	230
16540	16
16541	32
16542	1
16543	43
16544	14
16545	254
16546	6
16547	239
16548	237
16549	120
16550	230
16551	16
16552	32
16553	4
16554	17
16555	33
16556	0
16557	25
16558	14
16559	254
16560	6
16561	239
16562	237
16563	120

16564	230
16565	6
16566	32
16567	4
16568	17
16569	223
16570	255
16571	25
16572	14
16573	254
16574	6
16575	239
16576	237
16577	120
16578	230
16579	4
16580	32
16581	1
16582	35
16583	126
16584	254
16585	128
16586	40
16587	182
16588	126
16589	254
16590	23
16591	200
16592	54
16593	52
16594	195
16595	141
16596	64

COLLECTION E.T.S.F. MICRO-SYSTEMES

1 - A. VILLARD et M. MIAUX, *Un microprocesseur pas à pas*
2 - A. VILLARD et M. MIAUX, *Systèmes à microprocesseur*
3 - P. GUEULLE, *Maîtrisez votre ZX 81*
4 - E. FLOEGEL, *Du Basic au Pascal*
5 - P. COURBIER, *Vous avez dit Basic ?*
6 - M. MARCHAND, *Vous avez dit micro ?*
7 - P. GUEULLE, *Pilotez votre ZX 81*
8 - M. JACQUELIN, *La micro-informatique et son A-B-C*
9 - M. OURY, *Maîtrisez les TO 7 et TO 7-70*
10 - P. GUEULLE, *Pilotez votre Oric 1 + Atmos*
11 - P. JOUVELOT et D. LE CONTE DES FLORIS, *Système d'exploitation et logiciel de base des micro-ordinateurs*
12 - P. GUEULLE, *Robotisez votre ZX 81*
13 - M. CAUT, *J'apprends le Basic*
14 - C. MALOSSE, C. TASSET, P. PRUT, *La micro, c'est pas sorcier !*
15 - R. GREGOIRE, *Bus IEEE*
16 - M. OURY, *Maîtrisez le MO5*
17 - P. GUEULLE, *Votre ordinateur et la télématique*
18 - P. COURBIER, *Connaissez-vous Macintosh ?*
19 - M. ROUSSELET, *Graphismes en kits*
20 - S. ARQUIE, *Micro-informatique et P.M.E.*
21 - R. SCHULZ, *60 solutions pour Oric 1 et Atmos*
22 - H. HUNIC, *Listes et tableaux numériques en Basic*
23 - C. TAVERNIER, *Les secrets du Minitel*
24 - P. COURBIER, *Macintosh, quels logiciels ?*
25 - P. GUEULLE, *Faites de l'argent avec votre micro*
26 - R. COHEN, *Parlez-vous dBase II ?*
27 - G. MICLOT, *Compta sur TO 7-70*
28 - M. JACQUELIN, *La micro et ses langages*
29 - C. TAVERNIER, *Maîtrisez votre EXL 100*
30 - J.L. TERRASSON, *Les périphériques des micro-ordinateurs*
31 - X. LEROY, *Logo, langage pour tous*
32 - G. BOSSUET, *Dictionnaire Logo*
33 - B. LOUBEAU, *Fichiers séquentiels sur micro-ordinateur*
34 - M. ROUSSELET, *Graphisme 3D*
35 - M. OURY, *Robotisez les TO 7 et MO5*

Un livre de 128 pages, format 11,7 × 16,5 cm

50 PROGRAMMES POUR ZX 81

Utiles ou divertissants, les programmes qui sont rassemblés dans cet ouvrage sont originaux et utilisent au mieux toutes les fonctions du ZX 81. Ils sont tous écrits pour la version de base avec **mémoire RAM de 1 K.** Loin d'être limités, ils constituent au contraire un **exercice très profitable** pour apprendre à ne pas dépasser la place mémoire disponible.
Votre propre imagination et quelques idées glanées dans ces lignes vous permettront de créer très rapidement des programmes personnels.

Quelques programmes :

- **Aide à la programmation**
- **Conversion de température**
- **Conversion décimal en binaire**
- **Conversion binaire en décimal**
- **Calcul des intérêts**
- **Conjugaison**
- **Loto**
- **Billard**
- **Le champ de mines**

Edité par :

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue - 75940 PARIS CEDEX 19

Achevé d'imprimer
sur les presses
de l'Imprimerie Marcel Bon
70001 VESOUL
Dépôt légal : juin 1985
N° d'éditeur : 467
N° d'imprimeur : 2922

MICRO SYSTEMES

ETSF

PROGRAMMER EN LANGAGE MACHINE ET JOUER SUR ZX 81

Cet ouvrage s'adresse à tous les possesseurs de ZX 81 équipé de la mémoire 16 K. Son but est de permettre à ceux qui n'ont aucune notion de **langage machine** de découvrir progressivement les principales connaissances nécessaires à sa programmation.
Les instructions les plus utiles sont répertoriées dans un aide-mémoire très pratique. De plus, cinq **programmes originaux** vous feront apprécier les possibilités de ce langage sur ZX 81.

Les programmes :

- Zigzag
- Chronomètre digital géant
- Dédale
- Casse-briques
- Défense spatiale

9 782855 351100

ISBN 2-85535-110-3

TARIF CODE B

20

PROGRAMMER EN LANGAGE MACHINE
ET JOUER SUR ZX 81

G. ISABEL
B. N'GUYEN VAN TINH